Design, Implementation and Deployment of a Secure Account-Based Electronic Payment System

Mihir Bellare, Juan A. Garay[*], Ralf Hauser, Amir Herzberg, Hugo Krawczyk,

Michael Steiner, Gene Tsudik, Els Van Herreweghen, Michael Waidner[†]

Abstract

This paper discusses the design, implementation and deployment of a secure and practical payment system for electronic commerce on the Internet. The system is based on the iKP family of protocols – iKP ($i = 1, 2, 3$) – developed at IBM Research. The protocols implement credit card-based transactions between buyers and merchants while the existing financial network is used for payment clearing and authorization. The protocols are extensible and can be readily applied to other account-based payment model, such as debit cards. They are based on careful and minimal use of public-key cryptography and can be implemented in either software or hardware. Individual protocols differ in both complexity and degree of security.

In addition to being both a pre-cursor and a direct ancestor of the well-known SET standard, iKP-based payment systems have been in continuous operation on the Internet since mid-1996. This longevity – as well as the security and relative simplicity of the underlying mechanisms – make our experience with iKP unique. For this reason, this paper also reports on, and addresses, a number of practical issues arising in the course of implementation and real-world deployment of a secure payment system.

1 Introduction and Overview

At this day and age it is hardly necessary to justify, or stress the importance of, electronic commerce. Suffice it to say that it has been rapidly gaining momentum since early nineties, and has been equally appealing to on-line merchants, consumers and payment providers.

There is a widespread agreement that to enable electronic commerce one needs the means for *secure electronic payments.* Indeed, the appeal of electronic commerce without electronic payment is limited. Moreover, *insecure* electronic payment methods are more likely to impede, than to promote, electronic commerce. Thus we begin with the premise that security for electronic payment is of the utmost importance.

In this paper we present and discuss a family of secure electronic payment protocols – iKP (i-Key-Protocol, $i = 1, 2, 3$). These protocols are compatible with the existing card-based business models and payment system infrastructure. They involve three parties: the buyer (who makes the actual payment), the merchant (who will receive the payment) and the acquirer gateway (who acts as an intermediary between the electronic payment world and the existing payment infrastructure, and authorizes transactions by using the latter). Hereafter, we will refer to the acquirer gateway as simply *the acquirer.*

Within this framework we focus on the credit card payment model since it has been the most popular thus far and likely to remain so in the near future. However, note that other account-based payment models such as debit cards don't really differ from the credit card model from a technical viewpoint, and are easily supported by iKP.

[*]Contact author: Information Sciences Research Center, Bell Labs – Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07093. Phone: (908) 582-5867, FX: (908) 582-1239. E-mail: garay@research.bell-labs.com.

[†]Work was done while all authors were with the IBM Research Division.

All iKP protocols are based on public-key cryptography, but they vary in the number of parties (out of the three involved) that possess individual public key pairs and hence can create digital signatures. This number is reflected in the name of the individual protocols: 1KP, 2KP, and 3KP. The iKP protocols offer increasing levels of security and sophistication as the number of parties who possess own public key pairs increases.

The simplest protocol, 1KP, requires only the acquirer to possess a public key-pair. Buyers and merchants only need to have authentic copies of the acquirer's public key, reflected in a public key certificate. This involves a minimal public key infrastructure (PKI) to provide certificates for a small number of entities, namely, the acquirers. Such an infrastructure can be operated, for example, by a large credit card company. In the 1KP setting, buyers are authenticated on the basis of their credit card numbers, and possibly associated secret PINs. Payments are authenticated by communicating the credit card number and PIN, appropriately encrypted under the acquirer's public key, and properly bound to relevant information (purchase amount, identities, etc.). This prevents fraudulent merchants from collecting creditcard numbers and creating phony payments.[1] 1KP does not offer non-repudiation for messages sent by buyers and merchants. This means that disputes about the authenticity of payment orders are not unambiguoulsy resolvable within the digital system.[2]

2KP demands that merchants, in addition to acquirers, hold public key-pairs and public key certificates. The protocol can then provide non-repudiation for messages originated by merchants. Additionally, 2KP enables buyers to verify that they are dealing with *bona fide* merchants by checking their certificates, without any on-line contact with a third party. As in 1KP, payment orders are authenticated via the buyer's credit card number and PIN, encrypted before transmission.

3KP further assumes that buyers have their own public key-pairs and public key certificates, and thus it achieves non-repudiation for all messages of all parties involved. Payment orders are authenticated by the combination of credit card number, optionally a PIN, and a digital signature of the buyer. This makes the forging of payment orders computationally infeasible. Additionally, 3KP enables merchants to authenticate buyers on-line. This requires a full public key infrastructure covering all parties involved.

The reason for designing these three variants was to enable gradual deployment: 1KP requires only a minimal PKI and would have been suitable for immediate deployment at the time it was proposed. 2KP requires a PKI covering all merchants, 3KP one covering all merchants and all card holders. The way how iKP and its successor, SET, are deployed showed that there was actually no need for 1KP.

All iKP protocols can be implemented in either software or hardware. In fact, in 1KP and 2KP the buyer does not even need a personalized payment device: only credit card data and the PIN (if present) must be entered to complete a payment. However, for the sake of increased security, it is obviously desirable to use a tamper-resistant device to protect the PIN and – in case of 3KP – the secret key of the buyer.

We emphasize that the goal of iKP is to enable *payments*. It is not concerned with any aspect of the determination of the order; it assumes that the order, including price, have already been decided on between buyer and merchant. It does, however, securely link order information into the payment to enable effective dispute handling.

The iKP protocols do not explicitly provide encryption of the order information. Such protection is assumed to be provided by other existing mechanisms, e.g., SSL [FKK96]. The decoupling of order encryption from the electronic payment protocol is an important design principle of iKP which supports compatibility with different underlying browsing and privacy-protecting mechanisms. It also adds to the simplicity, modularity, and ease of analysis of the protocols. An additional advantage is freeing iKP from US export restrictions related to the use of bulk encryption. Nonetheless, if desired, the iKP family (especially, 2KP and 3KP) can be easily extended to generate shared keys between buyer and merchant for protection of browsing and order information.

The rest of this paper is organized as follows: Section 2 provides a brief summary of the history of iKP and its relation to the current credit card payment standard, SET. The different roles – buyer, merchant,

[1] Strictly speaking, one cannot consider a number that is given to any restaurant waiter or receptionist a valuable secret, in any sense. But even if the buyer is not liable, knowing his or her credit card number is sufficient to commit certain frauds, and thus overall system security is improved if this number is protected.

[2] From a legal point of view such ambiguities are not necessarily a problem – provided there are fixed rules how to resolve them, and all parties are aware of these rules. Some consequences of systems where those rules were not appropriately designed are illustrated in [And94].

acquirer – are introduced in Section 3, and their different security requirements are analyzed in Section 4. The iKP family is described and analyzed in Section 5, and an implementation architecture for it is proposed in Section 6. The actually implemented protocols are specified in detail in Appendix A.

2 History and Related Work

iKP was developed in early 1995 by a group of researchers at the IBM Research labs in Yorktown Heights and Zürich. Right from the beginning our goal was to work towards an open industry standard. We distributed the iKP protocols in the Internet Draft form, invited comments from the scientific community, and presented our design at the Internet Engineering Task Force meeting in Summer of 1995. Subsequently iKP was incorporated into the "Secure Electronic Payment Protocols (SEPP)," a short-lived standardization effort by IBM, MasterCard, Europay and Netscape. SEPP, in turn, was a key starting point for "Secure Electronic Payments (SET)," the joint VISA/MasterCard standard for credit card payments [MV97]. In fact, SET still retains some of the iKP-esque features.

Other important ancestors of SET are the "CyberCash Credit Card Protocol" by CyberCash, and the "Secure Transaction Technology (STT)" by Microsoft and VISA. All these ancestors were proposed independently of each other but used more or less the same type of cryptographic protocols.

From 1994 to 1996 an almost countless number of payment protocols for all kinds of payment models were proposed (see [AJSW97] for a survey). One important difference between iKP and most of these proposals is that iKP was not just a paper design: The "Zurich iKP Prototype (ZiP)" is a fully operational prototype of 2KP and 3KP. Although ZiP never became a commercial product, it has been successfully deployed in a number of business trials in Europe and Japan since mid-1996. At the time of this writing, Interpay Nederland is still using ZiP in their I-Pay system, supporting 80 online merchants and 17000 card holders.

Another important difference between iKP and other credit card payment protocols is its simplicity and – to the extent the following term can be used – elegance: We designed iKP from a small, well-defined set of security requirements (see Section 4), which resulted in a multi-party secure scheme where no party is forced to trust other parties unnecessarily. We focused on the core payment functionality and deliberately omitted all non-payment functionality that could be easily added on top of iKP, such as secrecy of order information or fair delivery of goods. Finally we designed iKP as a family of protocols, allowing for a gradual deployment.

Today only two approaches for secure credit card payments over the Internet are practically relevant: SET, and encryption of credit card data via SSL [FKK96], respectively TLS [DA98].

SET and its ancestor iKP, in particular 3KP, are very similar. The main difference is in their complexity: iKP was designed as a lightweight protocol that provides the core payment functionality only, and is therefore relatively simple to understand and to analyze. SET was designed to support all options that exist in today's credit card operation and is therefore semantically much richer than iKP, but also much more difficult to analyze.

SSL is the *de facto* standard for secure (i.e., encrypted and integrity-protected) client-server communication in the world wide web, and is integrated in virtually all web browsers and servers. SSL uses public-key cryptography, like SET and iKP, but typically only servers (i.e., merchants) have public-key certificates while clients (i.e., buyers) are anonymous. Encrypting credit card data with SSL is certainly better than sending them in the clear, but the gain in payment security is very limited:

- For the acquirer the use of SSL is completely transparent – no messages are signed – and thus the merchant does not gain any security.
- SSL does not hide the credit card number or anything else from the merchant. Thus, it cannot be used for PIN-based authorization.
- Unlike SET or ZiP, SSL does not mandate any specific public-key infrastructure. Thus there is no guarantee that a buyer can verify the merchant's public-key certificate, and even if the certificate can be verified the semantics of such a certicicate is not clear.

Most types of payment systems, not just credit and debit cards, exist in the digital world. Typically each model requires its own type of protocols, i.e., one cannot expect that iKP can be applied to payment models

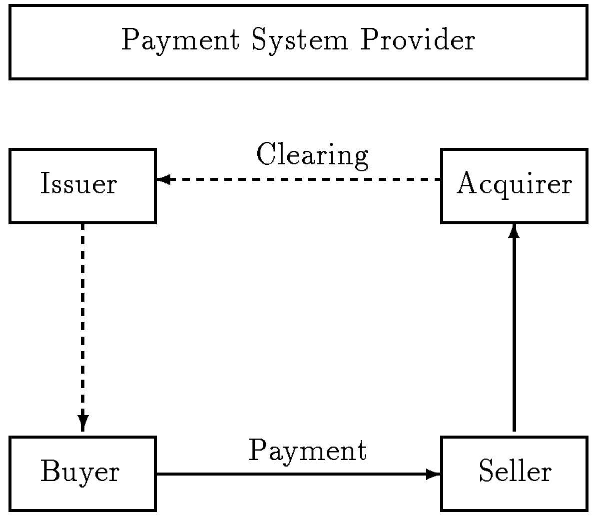

Figure 1: Generic model of a payment system

that are very different to the credit card model. We refer to [AJSW97] for a survey of other payment models and protocols.

3 Payment Model

PARTIES. All *i*KP protocols are based on the existing credit-card payment system. The parties in the payment system are shown in Figure 1.

The payment system is operated by a *payment system provider* has fixed business relations with certain banks who act as *issuers* of credit cards to buyers, or as *acquirers* of payment records from merchants (sellers). Each issuer has a Bank Identification Number, BIN, which it receives at the time it signs up with a payment system provider, and which is embossed on each credit card issued as part of the credit card number. The BIN also identifies the payment system provider.

A *buyer* receives a credit card from an issuer, and is in possession of a PIN as is common in current systems. In 1KP and 2KP, payments will be authenticated only by means of the credit card number and this PIN (both suitably encrypted!), while in 3KP a digital signature is used, in addition to the credit card number.

It is assumed (as can be expected for electronic payment) that the buyer is using a computer to execute the payment protocol. Since this computer must receive the buyer's PIN or secret signature key, it must be a trustworthy device. We caution that even a buyer-owned computer is vulnerable: it may be used by several people and it may contain a Trojan horse or a virus that could steal PINs and secret keys. The best payment device would be a secure isolated computer, e.g., a tamper-resistant smartcard, connected to the computer used for shopping via a buyer-owned smartcard reader with its own keyboard and display. (This is often called an *electronic wallet.*) Technically, 1KP and 2KP can be used with any kind of payment device, while for 3KP the buyers need personal devices that store their secret signature keys and certificates.

A *seller* signs up with the payment system provider and with a specific bank, called an *acquirer,* to accept deposits. Like a buyer, a seller needs a secure device that stores the seller's secret keys and performs the payment protocol.

Clearing between acquirers and issuers is done using the existing financial networks.

The *i*KP protocols deal with the *payment* transaction only (i.e., the solid lines in Figure 1), and therefore involve only three parties, called B – Buyer, S – Seller, and A – Acquirer (gateway). Recall that A is not the acquirer in the financial sense, but a *gateway* to the existing credit card clearing/authorization network. In other words, the function of A is to serve as a front-end to the *current infrastructure that remains unchanged.*

The protocols presented describe the core of a payment system only. Besides this, additional mechanisms are needed, e.g., for *providing statements of account.*

PUBLIC KEYS AND CERTIFICATION. Since all *i*KP protocols are based on public-key cryptography, we need a mechanism to authenticate these public keys. We assume a *certification authority, CA*, which has a secret key, SK_{CA}. Its public counterpart, PK_{CA}, is held by all other parties. CA will certify a public key of party X by signing the pair (X, PK_X) consisting of the identity of X and X's public key. (The signature is computed under SK_{CA}.) Note that PK_{CA} must be conveyed in an authenticated manner to every party. This will be typically done out-of-band, via any of a number of well-known mechanisms.

For simplicity's sake, the following discussions assume that there is only one certification authority. However it is easy to extend the protocols to support multiple certification authorities, e.g., such that the payment system provider at the top-level authority issues certificates to its constituent issuers and acquirers, while these, in turn, issue certificates to their buyers and sellers. The implementation described in Section 6.3 indeed supports this hierarchical model.

In all *i*KP protocols, an acquirer A has a secret key, SK_A, which enables signing and decryption. Its public counterpart, PK_A, (which enables signature verification and encryption) is held by each accredited seller together with its corresponding CA's certificate. As in current operation, acquirers receive the buyer's credit card numbers and PINs, and are trusted to keep these values confidential.

In 2KP, each seller, and in 3KP also each buyer, has a secret/public key-pair. They are denoted by $(\mathrm{SK}_S, \mathrm{PK}_S)$ and $(\mathrm{SK}_B, \mathrm{PK}_B)$, respectively. Both public keys are included in certificates issued by CA.

ADVERSARIES AND THREATS. We consider three different adversaries:

- **Eavesdropper** who listens to messages and tries to learn secrets (e.g., credit card numbers, PIN's)
- **Active attacker** who introduces forged messages in an attempt to cause the system to misbehave (e.g., to send him goods instead of to the buyer)
- **Insider** who either is some legitimate party or learns that party's secrets. (One example is a dishonest seller who tries to get paid without the buyer's authorization.)

Before listing the security requirements in Section 4, we briefly discuss common threats and attacks.

The Internet is a decentralized, heterogeneous network, without single ownership of the network resources and functions. In particular, one cannot exclude the possibility that messages between the legitimate parties would pass through a maliciously controlled computer. Furthermore, the routing mechanisms in the Internet are not designed to protect against malicious attacks. Therefore, it is folly to assume either confidentiality or authentication for messages sent over the Internet, unless proper cryptographic mechanisms are employed. To summarize, it is easy to steal information off the Internet. Therefore, at least credit card numbers and PINs must *not* be sent in the clear.

In addition, one must be concerned about the trustworthiness of the sellers providing Internet service. The kind of business that is expected in the Internet includes the so-called *cottage industry* – small sellers. It is very easy for an adversary to set up a shop and put up a fake electronic *storefront* in order to get buyers' credit card numbers. This implies that the credit card number should travel from buyer to seller without being revealed to the seller (who needs only the BIN which can be provided separately.)

Obviously, a good deal of care must be taken to protect the keys of acquirers. One of the biggest concerns is that of an adversary breaking into an acquirer computer through the Internet connection. Therefore, the acquirer's computer must be protected with the utmost care; including a very limited Internet connection using advanced firewall technology (e.g., [CB94, CGHK98].)

Furthermore, the trust in the acquirer's computer must be limited, so that a break-in would have a limited effect only.

4 Security Requirements

In this section we consider a range of potential requirements for each party involved in the payment process: issuer/acquirer, merchant, and buyer. They range from mandatory security requirements to optional features.

Issuer/Acquirer Requirements. The issuer and the acquirer are assumed to enjoy some degree of mutual trust. Moreover, an infrastructure enabling secure communication between these parties is already in place. Therefore, we join the requirements of the issuer and the acquirer.

A1– *Proof of Transaction Authorization by Buyer.* When the acquirer debits a certain credit card account by a certain amount, the acquirer must be in possession of an unforgeable *proof* that the owner of the credit card has authorized this payment. This proof must not be "replayable," or usable as proof for some other transaction. This means it must certify at least the amount, currency, goods description, seller identification, and delivery address, and be obtained in such a way that replay is not possible. (We use a combination of time stamps and nonces for this purpose). Note also that in this context the seller may be an adversary, and even such a seller must not be able to generate a fake debit. We distinguish between:

(a) *Weak Proof*, which authenticates the buyer to the acquirer but does not serve as a proof for third parties, and

(b) *Undeniable Proof*, which provides full non-repudiation, i.e., can be used to resolve disputes between the buyer and the payment system provider.

The same distinction will be made for all subsequently required proofs of transaction.

A2– *Proof of Transaction Authorization by Seller.* When the acquirer authorizes a payment to a certain seller, the acquirer must be in possession of an unforgeable *proof* that this seller has asked that this payment be made to him.

Seller Requirements. We ask for two guarantees for the seller.

S1– *Proof of Transaction Authorization by Acquirer.* The seller needs an unforgeable proof that the acquirer has authorized the payment. This includes certification and authentication of the acquirer, so that the seller knows he is dealing with the real acquirer, and certification of the actual authorization information. Note that again the amount and currency, the time and date, and information to identify the transaction must be certified. We also distinguish between (a) *Weak proof* and (b) *undeniable proof*, which provides full non-repudiation.

S2– *Proof of Transaction Authorization by Buyer.* Even before the seller receives the transaction authorization from the acquirer, the seller might need an unforgeable proof that the buyer has authenticated it. Again we distinguish between (a) *Weak Proof* and (b) *Undeniable Proof.* This requirement is necessary to provide for *off-line authorization.*

Buyer Requirements. We ask for the following guarantees to the buyer who is making the payment.

B1– *Unauthorized Payment is Impossible.* It must not be possible to charge something to a buyer's credit card without possession of the credit card number, PIN, and in case of 3KP, the buyer's secret signature key. Thus, neither Internet rogues nor malicious sellers must be able to generate spurious transactions which end up approved by the acquirer. This must remain the case even if the buyer has engaged in many prior legitimate transactions. In other words, information sent in one (legitimate) transaction must not enable a later spurious transaction. So in particular the PIN must not be sent in the clear, and not even be subject to guessing attacks! Similar to the two type of proofs of transactions, we distinguish between:

(a) *Impossibility*, which means that unauthorized payments are impossible provided the acquirer is honest and its secret key is not available to the adversary, and

(b) *Disputability*, which means that even if the acquirer's secret key is available to the adversary (e.g., because the adversary co-operates with an insider), the buyer can prove that he/she did not authorize the payment.

In fact, these two requirements are typically met by meeting the corresponding acquirer requirements A1.a and A1.b, respectively.

B2– *Proof of Transaction Authorization by Acquirer.* The buyer would like to be in possession of proof that the acquirer authorized the transaction. This "receipt" from the acquirer is not of paramount importance, but is convenient to have. Again, we distinguish between (a) *Weak Proof* and (b) *Undeniable Proof* (full non-repudiation).

B3– *Certification and Authentication of Seller.* The buyer needs a proof that the seller is accredited at an acquirer (which could be considered as some guarantee for the trustworthiness of the seller).

B4– *Receipt from Seller.* The buyer wants a proof that the seller who has made the offer has received payment and promised to deliver the goods. This takes the form of an undeniable receipt. 2KP and 3KP will satisfy this requirement, but will not ensure *fairness*[ASW98, Aso98]: The seller can always refuse sending this receipt while already having received the authorization message from the acquirer. In this case, the buyer must take the next statement of account as a replacement for this receipt.

ADDITIONAL POSSIBLE BUYER REQUIREMENTS. The following requirements (B5 – B6) may also be desirable. Here we shortly discuss their relation to *i*KP; however, they are not explicitly addressed by the *i*KP protocols.

B5– *Privacy.* Buyers want privacy of their order and payment information. For example a businessman may be purchasing the latest information on certain stocks and may not want competitors to know which stocks he is interested in. The privacy of order information and amount of payment should be implemented independently of the payment protocol, e.g., based on SSL [FKK96]. *i*KP does provide some privacy: it does not reveal order information to any other party than the seller, at least as long as there is no dispute. But it does not include encryption of these data. Obviously, credit card number and PIN must be protected carefully, which is achieved within *i*KP by encrypting them with the acquirer's public key. (This is the only application of *encryption* in *i*KP, which is made in order to facilitate exportability from the US.)

B6– *Anonymity.* Besides confidentiality of order and payment information, buyers may want *anonymity* from eavesdroppers and (optionally) also from the seller. It is also conceivable that the buyer may even want anonymity with respect to the payment system provider.

*i*KP doesn't focus on anonymity and in particular does offer no anonymity from the payment system provider. This might be desirable for systems that aim to imitate cash, but is not essential for protocols, like *i*KP, that follow the credit card-based payment model. *i*KP tries however to minimize the exposure of the buyers identity towards the outside and the seller.

5 The *i*KP Protocol Family

In this section we present the three *i*KP protocols, $i \in \{1, 2, 3\}$. We first describe the cryptographic primitives used by the protocols, as well as their general structure. In the presentation, enough information (e.g., atomic and composite fields) is included in the protocol flows so as to show how the protocols satisfy the security requirements listed in the previous section. A complete description of this information is given in the protocol specification section, Section 6.

PRIMITIVES AND KEYS. Figure 2 summarizes the notation for the cryptographic keys held by the various parties, and the cryptographic primitives we will be using. While A's key pair must enable signature and encryption, all other key pairs need to enable signatures only.[3] Note that signing and encryption are *independent* operations; in particular, $\mathcal{E}_X(\mathcal{S}_X(\alpha)) \neq \alpha$.

We want an encryption function $\mathcal{E}_X$ which provides some form of "message integrity." Decryption of a ciphertext results either in a plaintext message, or in a flag indicating non-validity. Formally, the primitive we want is an encryption function which is secure against adaptive choosen ciphertext attacks. This means that correct decryption convinces the decryptor that the transmitter "knows" the plaintext that was encrypted.

[3] For simplicity of exposition, we will be assuming in this section that each party has only one pair of keys; in Section 6 we will be using dedicated pairs for encryption and signature.

- **Keys:**

$\mathrm{PK}_X, \mathrm{SK}_X$	Public and secret key of Party X (X = Certification Authority CA, Buyer B, Seller S, Acquirer A).
CERT_X	Public key certificate of Party X, issued by CA. We assume it includes X, PK_X and CA's signature on X, PK_X.

All protocols assume A has a public key, and any party needing it has PK_{CA}. 1KP assumes no other keys; 2KP additionally assumes S has a public key; 3KP further assumes B also has a public key.

- **Cryptographic primitives:**

$\mathcal{H}(\cdot)$	A strong collision-resistant one-way hash function. Think of $\mathcal{H}(\cdot)$ as returning "random" values. (Examples: MD5 [Riv92], SHA-1 [NIS95])
$\mathcal{H}_k(K, \cdot)$	This one-way hash function requires in addition to collision-resistance that no information is leaked about the other arguments if its first argument K is choosen at random. I.e., $\mathcal{H}_k(K, \cdot)$ should behave like a family of pseudo-random functions. (Examples: HMAC [BCK96, Kra99]).
$\mathcal{E}_X(\cdot)$	Public-key encryption using PK_X, done in a way to provide not only confidentiality but also some kind of "message integrity."
$\mathcal{S}_X(\cdot)$	Signature with respect to SK_X. Note the signature of message M does NOT include M. We assume the signature function hashes the message before signing.

Figure 2: *Keys and cryptograhic primitives used in iKP protocols*

In particular, tampering with ciphertext is detectable. Two practical yet proveably secure schemes to achieve this are Optimal Asymmetric Encryption Padding (OAEP) [BR94] and Cramer-Shoup [CS98].

We stress that such encryption does not provide authentication in the manner of a signature, i.e., it does not provide non-repudiation. But it can be made to provide an authentication-like capability between parties sharing a key (such as the BAN or PIN).

We note that the encryption function is necessarily *randomized:* $\mathcal{E}_X$ invoked upon message m will use, to compute its output, some randomizer, so that each encryption is different from previous ones.

The prototype implementation described in Section 6 uses RSA with key length 1024 for signature, and for the basis of the plaintext aware encryption; it also uses MD5 as a hash function.

Figure 3 is a list of quantities that will occur in the protocols. Their meaning and usage will be further explained as we go along.

FRAMEWORK OF *i*KP PROTOCOLS. The protocols have a common framework. Figure 4 illustrates the flows at a very high level. Before the protocol begins, each party $X \in \{A, B, S\}$ has some starting information represented by $\mathrm{ST\text{-}INF}_X$. The buyer starts with the public key PK_{CA} of the certification authority. The seller has the certificate CERT_A of the acquirer, and the acquirer has his own certificate CERT_A plus the corresponding secret key SK_A. They may each also have other information, which differs depending on whether we are in 1KP, 2KP or 3KP, and will be specified at the appropriate time.

It is assumed that before the protocol starts, the buyer and the seller have agreed on the description and price of the items to buy. The functionality required to shop and agree on the item and price are to be provided by other means (e.g., the browser), not by *i*KP.[4] Thus, DESC and PRICE are part of the starting information of seller and buyer.

[4] Indeed, the consensus nowadays is that functions that are beyond payment, such as price negotiation, should be separated, and standardized; e.g., JEPI [CD97], SEMPER [Wai96].

- **Quantities occuring in all three protocols:**

SALT_B	Random number generated by B. Used to salt DESC and thus ensure privacy of order information (DESC) on the S to A link; also used to provide freshness of signatures (Sig_S and Sig_A).
PRICE	Amount and currency
DATE	Seller's date/time stamp, used for "coarse grained" replay protection of a payment
NONCE_S	Seller's nonce (random number) used for more "fine grained" replay protection of a payment
ID_S	Seller id. This identifies seller to acquirer.
TID_S	Transaction ID. This is an identifier chosen by the seller which uniquely identifies the context.
DESC	Description of purchase/goods, and delivery address. Includes payment information such as credit card name, bank identification number, and currency. Defines the agreement between buyer and seller as to what is being paid for in this payment transaction.
BAN	Buyer's Account Number (e.g., credit card no.). Includes expiration date.
R_B	Random number chosen by buyer to form ID_B. It must be random (not just unique) in order to serve as proof by the buyer that the seller agreed to the payment.
ID_B	A buyer pseudo-ID which $\text{ID}_B = \mathcal{H}_k(R_B, \text{BAN})$.
Y/N	Response from the clearing network: YES/NO or authorization code.
Text_j	For $j = 0, 1, 2, \ldots$. This is optional information that can accompany the flows. For example, can be used to carry context identifiers.

- **Quantities occuring in some of the protocols:**

PIN	Buyer PIN which, if present, can optionally be used in 1KP and 2KP to enhance the security.
V	Random number generated by seller in 2KP and 3KP for use as a proof that seller has accepted payment (i.e., to bind Confirm and Invoice messages).

Figure 3: *Definitions of atomic fields used in iKP protocols*

The basic protocol consists of five flows.[5] The exact content of these flows depends on the protocol: they are different in 1KP, 2KP and 3KP. At a high level, however, there is a common structure. The buyer starts with an Initiate flow. The seller responds by providing the Invoice. The buyer then makes the Payment which the seller uses to send an authorization request Auth-Request to the acquirer. The acquirer goes through the financial network to obtain the authorization and returns an authorization response Auth-Response to the seller. The latter processes this to produce a confirmation flow Confirm for the buyer.

The main difference between 1KP, 2KP and 3KP is the increasing use of digital signatures as more of the parties involved possess a public/secret key pair.

[5] Later, in Section 6, we will be adding some additional flows to provide for the cancellation of a transaction, payment clearance, and inquiry of status of a specific payment.

5.1 1KP

Protocol 1KP, illustrated in Figure 5, represents the initial step in the gradual introduction of a public-key infrastructure. Although it requires the use of public-key encryption by all parties, only the acquirer, A, needs to possess and distribute its own public key certificate, CERT_A. In particular, the total number of certificates to be issued by the certification authority is small as it depends only on the number of acquirers.

Like all members of the iKP Family, 1KP requires that all buyers and sellers have an authentic copy of PK_{CA}, the public key of the certification authority. A buyer B has an account number BAN (e.g., a credit card number) known to the acquirer. It may also have a secret PIN which is also known to the payment system (but not to the sellers!). Every seller has to know the certificate of the corresponding acquirer, CERT_A.

1KP does not require A to keep a state per buyer. Instead, the buyer's PIN is verified using the existing authorization infrastructure (which uses tamper-resistant technology for processing and verification of PIN's).

All parties in 1KP must perform certain public key computations. Encryption is only applied *once,* for sending account data (and PIN) from the buyer to the acquirer securely. Therefore, public key encryption is required from B only, while decryption is required from A only (this is true also for 2KP and 3KP). In 1KP, only A has to sign some data, which must be verified by B and S. We now provide the flow by flow actions of the parties. In the protocol, Common is the information held in common by all parties, and Clear is the information transmitted in the clear.

Initiate: The buyer forms ID_B by generating random number R_B and computing $\mathrm{ID}_B = \mathcal{H}_k(R_B, \mathrm{BAN})$. Generates another random number SALT_B to be used for "salting" the hash of merchandise description (DESC) in subsequent flows. Sets Text_0 to include desired protocol options (if any) and/or DESC. Sends Initiate.

Invoice: The computation of the second flow, Invoice, takes place as follows. The seller retrieves SALT_B and ID_B from Initiate. Chooses/obtains DATE—this is a time stamp, and indicates, say the hour as well. Generates nonce NONCE_S. The combination of DATE and NONCE_S will be used later by A to uniquely identify this order: the nonce disambiguates payments with a common DATE. Chooses transaction id TID_S which identifies the context. Computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$. Forms Common as defined above and computes $\mathcal{H}(\mathrm{Common})$. (Note: Seller does not need to additionally "salt" $\mathcal{H}(\mathrm{Common})$ because it contains the already-salted $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$.) Composes Text_1. (If B did not already have CERT_A then it could go here. Or this could include a context pointer for the buyer.) Finally sends Invoice.

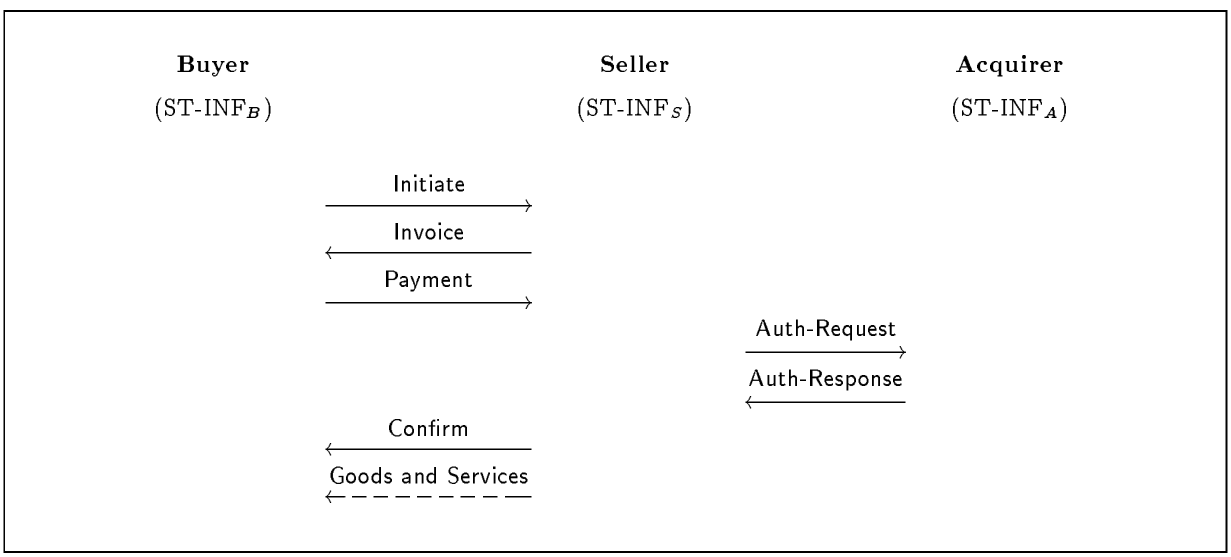

Figure 4: *Framework of iKP protocols*

- **Composite Fields:**

Common	PRICE, ID_S, TID_S, DATE, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$
Clear	ID_S, TID_S, DATE, NONCE_S, $\mathcal{H}(\mathrm{Common})$
SLIP	PRICE, $\mathcal{H}(\mathrm{Common})$, BAN, R_B, [PIN]
EncSlip	$\mathcal{E}_A(\mathrm{SLIP})$

- **Starting information of parties:**

$\mathrm{ST\text{-}INF}_B$	$\mathrm{DESC}, \mathrm{BAN}, \mathrm{PK}_{CA}, [\mathrm{PIN}]$
$\mathrm{ST\text{-}INF}_S$	$\mathrm{DESC}, \mathrm{PK}_{CA}, \mathrm{CERT}_A$
$\mathrm{ST\text{-}INF}_A$	$\mathrm{SK}_A, \mathrm{CERT}_A$

- **Protocol Flows:**

Initiate: $\quad B \xrightarrow{\mathrm{SALT}_B,\ \mathrm{ID}_B,\ [\mathrm{Text}_0]} S$

Invoice: $\quad B \xleftarrow{\overbrace{\mathrm{ID}_S,\ \mathrm{TID}_S,\ \mathrm{DATE},\ \mathrm{NONCE}_S,\ \mathcal{H}(\mathrm{Common})}^{\mathrm{Clear}},\ [\mathrm{Text}_1]} S$

Payment: $\quad B \xrightarrow{\overbrace{\mathcal{E}_A(\mathrm{SLIP})}^{\mathrm{EncSlip}},\ [\mathrm{Text}_2]} S$

Auth-Request: $\quad S \xrightarrow{\mathrm{Clear},\ \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}),\ \mathrm{EncSlip},\ [\mathrm{Text}_3]} A$

Auth-Response: $\quad S \xleftarrow{\mathrm{Y/N},\ \overbrace{\mathcal{S}_A(\mathrm{Y/N}, \mathcal{H}(\mathrm{Common}))}^{\mathrm{Sig}_A},\ [\mathrm{Text}_4]} A$

Confirm: $\quad B \xleftarrow{\mathrm{Y/N},\ \mathrm{Sig}_A,\ \mathrm{CERT}_A,\ [\mathrm{Text}_5]} S$

Figure 5: 1KP Protocol

Payment: Buyer retrieves Clear from Invoice. He retrieves ID_S, DATE, TID_S and NONCE_S. He validates DATE within a certain time skew. He computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$; he already has PRICE and ID_B, so that he can now form Common. He computes $\mathcal{H}(\mathrm{Common})$ and checks that this matches the value in Clear. He then forms the SLIP as defined in Figure 5. (It includes the price, the buyer's account number (credit card number), and $\mathcal{H}(\mathrm{Common})$. It also includes the salt R_B used to form the ID_B, and optionally the PIN if present.) The slip is now encrypted under the acquirer public key: he sets EncSlip $= \mathcal{E}_A(\mathrm{SLIP})$. This, along with the optional Text_2, is the Payment flow sent to the seller.

Auth-Request: The seller will now ask that the acquirer authorizes the payment. He forwards EncSlip. He also sends Clear and $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, and optional Text_3.

Auth-Response: The acquirer extracts Clear, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ and EncSlip from Auth-Request. It then does the following:

(1) Extracts from Clear the following– ID_S, TID_S, DATE, NONCE_S and the value h_1 which is supposed to be $\mathcal{H}(\mathrm{Common})$. It now checks for *replays*. That is, it makes sure that there is no previously processed request with these values of ID_S, TID_S, DATE and NONCE_S.

(2) Now it decrypts EncSlip. If the decryption fails, then the alteration of EncSlip (by an adversary or by S) is detected and the transaction is invalid. If not, A gets SLIP. Now A extracts PRICE, the value h_2 which is supposed to be $\mathcal{H}(\mathrm{Common})$, BAN, R_B, and, if present, the PIN from SLIP.

(3) It checks that $h_1 = h_2$—this ensures that buyer and seller agree on the order information (price, identity of seller, etc).

(4) It re-forms Common. (It has PRICE from SLIP. It has ID_S, TID_S, DATE, and NONCE_S from Clear. It can compute $\mathrm{ID}_B = \mathcal{H}_k(R_B, \mathrm{BAN})$ because it has R_B and BAN from SLIP. Finally it has $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ from Auth-Request. These put together yield Common.) It then computes $\mathcal{H}(\mathrm{Common})$ and checks this equals the value $h_1 (= h_2)$ above.

(5) Now it uses the credit card organization's existing clearing and authorization system to on-line authorize the payment: for this, it will forward BAN, PIN if present, the price, etc., as dictated by the authorization system. Upon receipt of a response Y/N from the authorization system, A computes a signature, using the function $\mathcal{S}_A$, on Y/N and $\mathcal{H}(\mathrm{Common})$.

Finally it sends Auth-Response and possibly Text_4. The latter could include TID_S so that the seller can easily recover the context.

Confirm: The seller receives Auth-Response. He extracts Y/N and the acquirer signature. He already has $\mathcal{H}(\mathrm{Common})$. Now he checks that the acquirer sent a valid signature of Y/N, $\mathcal{H}(\mathrm{Common})$. He then forwards Y/N to the buyer. He also forwards the acquirer signature so that the buyer may check it.

There are some final checks by the buyer: for example, he may want to check the acquirer's signature. We stress here that the use of $\mathcal{H}(\mathrm{Common})$ in the signature (as opposed to using the explicit values amount, currency, etc.), is done in order to protect the privacy of these data when transmitted to seller and buyer. We now look at which requirements 1KP satisfies.

A1(a) *Proof of Transaction Authorization by Buyer.* SLIP includes the BAN and the PIN. (The latter, if present, is known only to the buyer and payment system and is the basis of the security. If it is not present, one must assume the BAN is not known to an adversary.) Since B knows PK_{CA} and verifies CERT_A, it is ensured that B does not unwittingly send the BAN and PIN to a non-authorized party. A decrypts and checks that the BAN and PIN are correct. The plaintext-awareness of the encryption (see beginning of Section 5) implies that SLIP originated with the BAN and PIN holder. An adversary not knowing the BAN or PIN can neither create a fake SLIP nor modify the encryption of a legitimate one to its advantage.

Replay of a SLIP by a dishonest seller will be detected by the combination of the DATE and NONCE_M. There is an "acceptable delay" period T_{delay}. Slips containing a particular DATE are kept until for T_{delay} more time than that indicated by DATE. (For example, DATE could be the date and hour, and the delay period a day, meaning slips are kept for a day more than the DATE marked on them.) Within a particular value of DATE, different slips are disambiguated by the nonces.

The "semantic security" of the encryption (and in particular the fact that it is randomized) implies also security against *dictionary-attacks*. If the attacker knows all data in SLIP except PIN, he could compute encryptions $\mathcal{E}_A$(SLIP) for *all* possible values of PIN. With a deterministic encryption function, he could easily determine the correct PIN by comparing all encryptions with the one produced by C. Therefore, plain-text aware encryption is randomized: if SLIP is encrypted twice, two *different* cyphertexts are produced, which excludes this type of attack.

Note that PIN-based authentication provides a weak proof only. Signature-based authentication as used in 3KP provides an undeniable proof. Moreover, the probability of guessing the correct PIN is much higher than the probability of guessing a valid-looking signature.

It is important to stress that the "transaction" of which we want a proof includes the item description, and in particular the delivery address. It should not be possible for an adversary to divert a legitimate payment by changing the delivery address. The inclusion of $\mathcal{H}$(Common) in A's authorization is to prevent such attacks. In particular it prevents a certain kind of *person-in-the-middle* attack that we now describe.

An attacker that impersonates a seller can get the agreement of the buyer to buy something for a given amount. The adversary gets from the buyer an encrypted slip authorizing the payment. The adversary now impersonates the buyer to the seller, but this time the adversary buys for the same amount a (possibly) different merchandise with different delivery address and "pays" for it with the buyer's slip. Notice, however, that in this case there will be a mismatch between the view of the "order" by the real buyer and the sller, and, consequently, a mismatch in the value of $\mathcal{H}$(Common).

S1(b): *Proof of Transaction Authorization by Acquirer.* The unforgeable, undeniable proof is the digitally-signed message sent by A. Notice that we have used a digital signature so that non-repudiability is provided. The inclusion of $\mathcal{H}$(Common) prevents the replay of authorization messages which would result in fake authorization of buyer's orders.

Since the seller knows Common in advance, the signature would indicate any tampering in the information sent from seller to acquirer, and any disagreement between buyer and seller on the payment data.

The inclusion of $\mathcal{H}$(Common) both in the buyer-generated SLIP and directly in Auth-Request by the seller enables A to detect a disagreement between seller and buyer with respect to the order contents (even before submitting the transaction to the clearing network).

B1(a): *Unauthorized Payment is Impossible.* This is a direct consequence of the achievement of A1(a).

B2(b): *Proof of Transaction Authorization by Acquirer.* As for **S1**(b).

B5: *Privacy.* Some partial privacy is provided. Specifically, the acquirer is not given DESC, but rather $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$. Furthermore, the acquirer, or an eavesdropper on the acquirer-to-seller link, cannot obtain DESC via a dictionary attack, as we now explain.

In a dictionary attack, the attacker has some small set of possible values of DESC, and want to see whether one of them is what the buyer is ordering. Had we not used the salt, but just sent $\mathcal{H}$(DESC), the attacker could easily make the check by evaluating $\mathcal{H}$ on his values and seeing whether one of the results matches the value $\mathcal{H}$(DESC) in the flow. But assuming he doesn't know SALT_B then by salting DESC in $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ all possible description DESC$'$ will will have the same likelihood due to the pseudo-random nature of $\mathcal{H}_k()$. Of course, if he was powerful enough to obtain SALT_B of the buyer-to-seller link (where it was transmitted in the clear) he would be able to do the dictionary attack, but that he can eavesdrop like that on both links is not too likely. Also, if privacy is really a concern, the buyer-to-seller communication may be protected by alternative means (e.g., SSL [FKK96]).

We stress that provision of privacy is not a primary concern of a payment protocol. However, we wish at least to not give anything away that should not be, and took the chance to add whatever privacy we could add without much cost.

The last flow from seller to buyer in which the signed authorization by the acquirer is transmitted is optional. It only serves as a receipt for the buyer but is not needed for the security of the payment protocol.

To summarize, 1KP is a simple and efficient protocol whose main achievement was (at the time of its design, circa 1995) to get a secure electronic payment system with as little modification as possible to the existing infrastructure. Its main weaknesses are: 1) the buyer authenticates itself via the acquirer and only using an account number and PIN (as opposed to a strong authentication via a digital signature);

2) the seller does not directly authenticate itself to the buyer or acquirer (there is some level of indirect authentication via the buyer's SLIP and the authorization by the acquirer); and 3) neither seller nor buyer provide undeniable receipts for the transaction. Upgrading 1KP to provide these missing features results in the protocols described in the next two subsections, namely, 2KP and 3KP.

- **Composite Fields:**

Common	PRICE, ID_S, TID_S, DATE, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, $\mathcal{H}(V)$
Clear	ID_S, TID_S, DATE, NONCE_S, $\mathcal{H}(V)$, $\mathcal{H}(\mathrm{Common})$
SLIP	PRICE, $\mathcal{H}(\mathrm{Common})$, BAN, R_B, [PIN]
EncSlip	$\mathcal{E}_A(\mathrm{SLIP})$
Sig_S	$\mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathcal{H}(V))$

- **Starting information of parties:**

$\mathrm{ST\text{-}INF}_B$	$\mathrm{DESC}, \mathrm{BAN}, \mathrm{PK}_{CA}$, [PIN]
$\mathrm{ST\text{-}INF}_S$	$\mathrm{DESC}, \mathrm{PK}_{CA}, \mathrm{CERT}_A, \mathrm{SK}_S, \mathrm{CERT}_S$
$\mathrm{ST\text{-}INF}_A$	$\mathrm{PK}_{CA}, \mathrm{SK}_A, \mathrm{CERT}_A$

- **Protocol Flows:**

Initiate: $\quad B \xrightarrow{\mathrm{SALT}_B,\ \mathrm{ID}_B,\ [\mathrm{Text}_0]} S$

Invoice: $\quad B \xleftarrow{\overbrace{\mathrm{ID}_S,\ \mathrm{TID}_S,\ \mathrm{DATE},\ \mathrm{NONCE}_S,\ \mathcal{H}(V),\ \mathcal{H}(\mathrm{Common})}^{\mathrm{Clear}},\ \overbrace{\mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathcal{H}(V))}^{\mathrm{Sig}_S},\ \mathrm{CERT}_S,\ [\mathrm{Text}_1]} S$

Payment: $\quad B \xrightarrow{\overbrace{\mathcal{E}_A(\mathrm{SLIP})}^{\mathrm{EncSlip}},\ [\mathrm{Text}_2]} S$

Auth-Request: $\quad S \xrightarrow{\mathrm{Clear},\ \mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC}),\ \mathrm{EncSlip},\ \mathrm{Sig}_S,\ \mathrm{CERT}_S,\ [\mathrm{Text}_3]} A$

Auth-Response: $\quad S \xleftarrow{\mathrm{Y/N},\ \overbrace{\mathcal{S}_A(\mathrm{Y/N}, \mathcal{H}(\mathrm{Common}))}^{\mathrm{Sig}_A},\ [\mathrm{Text}_4]} A$

Confirm: $\quad B \xleftarrow{\mathrm{Y/N},\ V,\ \mathrm{Sig}_A,\ \mathrm{CERT}_A,\ [\mathrm{Text}_5]} S$

Figure 6: 2KP Protocol

5.2 2KP

The second protocol, 2KP, is illustrated in Figure 6. The basic difference with respect to 1KP is that, in addition to A, each seller S needs to possess a public key with a matching secret key, and distribute its own public key, with its certificate, CERT_S.

We now describe the additions to the flows and actions. There are two new elements in Invoice. The first is that the seller chooses a random value V and puts $\mathcal{H}(V)$ in Invoice. (The inclusion of V in Confirm will later serves as a (one-time) "signature" thereby saving the seller one signature computation. See below.) This value will be added to Common for what follows. Second, the seller signs (using SK_S) the pair of strings $\mathcal{H}(\mathrm{Common})$ and $\mathcal{H}(V)$ and includes this signature Sig_S in Invoice too. Furthermore the seller includes CERT_S so that the buyer can check his signature. Upon receipt of Invoice the buyer checks the seller's signature, and then proceeds as before to generate Payment. Auth-Request is augmented by the seller to include the same signature Sig_S he sent to the buyer earlier, together with CERT_S. The acquirer checks this signature before authorizing payment. Finally, the value V is included by the seller in Confirm. The buyer computes $\mathcal{H}(V)$ and checks that it matches the value sent earlier in Invoice.

2KP satisfies all the requirements addressed by 1KP as well as:

A2: *Proof of Transaction Authorization by Seller.* This is achieved by the inclusion of the seller's signature Sig_S and certificate CERT_S, and the acquirer's verification of these.

B3: *Certification and Authentication of Seller.* Similarly achieved by inclusion of signature of seller and its check by buyer.

B4: *Receipt from Seller.* This is achieved by the combination of S's signed message sent to A, A's signed authorization message, and the value V sent in confirm. V assures the buyer (and any third party) that the seller has accepted the authorization response. (This is the payment if Y/N is yes, and the statement of rejection otherwise.) This is because no other party is capable of finding V. (It would require inverting the one-way function $\mathcal{H}$ on the point $\mathcal{H}(V)$.) Note it is important here that the buyer check Sig_A—else an adversary can flip Y/N after the seller sends Y/N and V. Thus, the combination of Sig_S, V and Sig_A give the buyer undeniable proof of the seller's agreement to the transaction's outcome (whether positive or negative). The same could be achieved by S signing A's authorization message, but at the cost of an additional signature.

Obviously, S can refuse forwarding A's authorization message to the buyer and sending its last message. In this case, B does not know whether the transaction was aborted or finalized (this must be handled based on the next statement of account).

5.3 3KP

As can be expected, in the last protocol—3KP—all protocol participants, including buyers, possess a public key, with the associated secret key and certificate. As illustrated in Figure 7, all parties are now able to provide non-repudiation.

The CERT_B sent to the seller may not only contain the buyer's public key and ID, but also further data. This further data is included in the certificate *in salted hashed form* using $\mathcal{H}_k()$. This allows to open the information only on demand and doesn't leak information to unauthorized users. For instance, CERT_B might include the hash of the buyer's physical address, and if ordered goods should be sent to B's home address, B can reveal "Buyer's physical address" and the corresponding salt to the seller who can verify it based on CERT_B. Similarly CERT_B can securely link the BAN to the signing key. This allows the acquirer to efficiently verify that the payer has the necessary authority over BAN contained in the SLIP. See the use of SALT_C in the Z*i*P protocols below and Krawczyk [Kra99] for a more indepth study of this issue.

The buyer's signature serves as undeniable proof of transaction (A1.b), and enables disputability (B1.b). On the other hand, the sellers can link all payments of the buyer with CERT_B and B's signature, i.e., the buyer loses some of the privacy compared to 1KP and 2KP. One way to avoid this is by encrypting CERT_B and the signature with A's public key.

Notice that in 3KP PIN numbers can still be used, but only for compatibility with the existing infrastructure. Except for that reason, PINs can be safely omitted since the level of authentication provided by

- **Composite Fields:**

Common	PRICE, ID_S, TID_S, DATE, NONCE_S, ID_B, $\mathcal{H}_k(\text{SALT}_B, \text{DESC})$, $\mathcal{H}(V)$
Clear	ID_S, TID_S, DATE, NONCE_S, $\mathcal{H}(V)$, $\mathcal{H}(\text{Common})$
SLIP	PRICE, $\mathcal{H}(\text{Common})$, BAN, R_B, [PIN]
EncSlip	$\mathcal{E}_A(\text{SLIP})$
Sig_S	$\mathcal{S}_S(\mathcal{H}(\text{Common}), \mathcal{H}(V))$
Sig_B	$\mathcal{S}_B(\text{EncSlip}, \mathcal{H}(\text{Common}))$

- **Starting information of parties:**

ST-INF_B	$\text{DESC}, \text{BAN}, \text{PK}_{CA}, \text{SK}_B, \text{CERT}_B, [\text{PIN}]$
ST-INF_S	$\text{DESC}, \text{PK}_{CA}, \text{CERT}_A, \text{SK}_S, \text{CERT}_S$
ST-INF_A	$\text{PK}_{CA}, \text{SK}_A, \text{CERT}_A$

- **Protocol Flows:**

Initiate: $\quad B \xrightarrow{\text{SALT}_B,\ \text{ID}_B,\ \text{CERT}_B,\ [\text{Text}_0]} S$

Invoice: $\quad B \xleftarrow[]{\overbrace{\text{ID}_S,\ \text{TID}_S,\ \text{DATE},\ \text{NONCE}_S,\ \mathcal{H}(V),\ \mathcal{H}(\text{Common})}^{\text{Clear}},\ \overbrace{\mathcal{S}_S(\mathcal{H}(\text{Common}), \mathcal{H}(V))}^{\text{Sig}_S},\ \text{CERT}_S,\ [\text{Text}_1]} S$

Payment: $\quad B \xrightarrow{\overbrace{\mathcal{E}_A(\text{SLIP})}^{\text{EncSlip}},\ \overbrace{\mathcal{S}_B(\text{EncSlip}, \mathcal{H}(\text{Common}))}^{\text{Sig}_B},\ [\text{Text}_2]} S$

Auth-Request: $\quad S \xrightarrow{\text{Clear},\ \mathcal{H}_k(\text{SALT}_B, \text{DESC}),\ \text{EncSlip},\ \text{Sig}_S,\ \text{CERT}_S,\ \text{Sig}_B,\ [\text{Text}_3]} A$

Auth-Response: $\quad S \xleftarrow{\text{Y/N},\ \overbrace{\mathcal{S}_A(\text{Y/N}, \mathcal{H}(\text{Common}))}^{\text{Sig}_A},\ [\text{Text}_4]} A$

Confirm: $\quad B \xleftarrow{\text{Y/N},\ V,\ \text{Sig}_A,\ \text{CERT}_A,\ [\text{Text}_5]} S$

Figure 7: 3KP Protocol

REQUIREMENTS/PROTOCOLS	1KP	2KP	3KP
Issuer/Acquirer			
A1. Proof of Transaction Authorization by Buyer	$\surd$	$\surd$	$\surd\surd$
A2. Proof of Transaction Authorization by Seller		$\surd\surd$	$\surd\surd$
Seller			
S1. Proof of Transaction Authorization by Acquirer	$\surd\surd$	$\surd\surd$	$\surd\surd$
S2. Proof of Transaction Authorization by Buyer			$\surd\surd$
Buyer			
B1. Unauthorized Payment is Impossible	$\surd$	$\surd$	$\surd\surd$
B2. Proof of Transaction Authorization by Acquirer	$\surd\surd$	$\surd\surd$	$\surd\surd$
B3. Certification and Authentication of Seller		$\surd\surd$	$\surd\surd$
B4. Receipt from Seller		$\surd\surd$	$\surd\surd$

Table 1: Comparison of the *i*KP payment protocols. A requirement marked by $\surd$ is satisfied but not disputable, while $\surd\surd$ indicates that the requirement is satisfied based on an undeniable proof, providing non-repudiation and disputability.

the buyer's signature is significantly superior to that provided by a PIN.

3KP satisfies all the requirements addressed by 2KP, as well as:

A1(b): *Undeniable Proof of Transaction Authorization by Buyer.* The buyer signs the SLIP using a secret key SK_B known to B only.

S2(b): *Proof of Transaction Authorization by Buyer.* Based on B's signature, S can verify that SLIP was signed by B. S cannot verify the correctness of the contents of SLIP, especially not of the PIN.

B1(b): *Unauthorized Payment is Impossible.* Follows from A1.b.

5.4 Comparison of the protocols

The *i*KP protocols presented above vary in the degree of both protection and complexity. They proceed in an incremental path towards electronic payment with strong security features with respect to all parties involved. Practically speaking, it was envisaged at the time of the design that 1KP would represent a short-term, interim step towards payment protocols with stronger security guarantees. Thereafter, 2KP and 3KP could be gradually phased in. Table 1 presents a comparison of the *i*KP protocols.

The *i*KP family can fulfill all stated requirements and, in particular, provide non-repudiatable receipts from the acquirer to the seller/buyer, and from the seller to the buyer. In case that the buyer also possesses a public-key pair (3KP), non-repudiation becomes possible also from the buyer to the seller/acquirer.

Anonymity is not a focus of *i*KP but 1KP and 2KP provides nevertheless complete anonymity of the buyer to the seller (and the outside): The buyer uses a pseudo-identity ID_B which is different in each transaction and therefore makeing the buyer not only unlinkeable but even untraceable. 3KP clearly leaks identity information through the certificates but through the use of pseudonyms buyers can stay at least unlinkable. Note also that special care has to be taken when hiding the BAN information in the certificate.

Order privacy against eavesdroppers could be achieved by applying a secure communication protocol (e.g., SSL [FKK96]), or, if desired, the *i*KP protocols themselves could be extended to provide that protection. Since *i*KP aims at credit-card-like payments, no anonymity against the payment system is provided.

As will be shown in the next section the *i*KP protocols can easily be extended to support batch processing of payments from the same buyer by the seller, or to guarantee amounts as commonly done, for example, in the case of car rentals. Another avenue for extensions are micro-payments: The relativly high cost of credit card transaction make *i*KP not directly suitable for payments of very small amounts. However, Hauser et al. [HSW96] show how *i*KP can be extended to support micro-payments without loosing strong multi-party security[6].

The *i*KP protocols (more specifically, 2KP and 3KP—or a combination thereof) were implemented at the IBM Zürich Research Lab. In the next chapter we turn to a more detailed description of the protocols as they were actually implemented and deployed.

6 Z*i*P: Implementation and Deployment

6.1 Protocol scenarios

The 2KP and 3KP protocols described in the previous section form the payment authorization core of the Z*i*P implementation. Additional functionality was added during design and implementation as a result of users' requests. This section focuses on the functionality of the implemented protocols; Appendix A contains their detailed specification.

The final Z*i*P protocol suite includes four protocol scenarios:

1. Payment Authorization (2KP and 3KP augmented with cancellation option)
2. Payment Clearance (Capture)[7]
3. Refunds
4. Inquiry

These sub-protocols are summarized below.

PAYMENT AUTHORIZATION. This is the basic payment scenario described in Section 5 and Figure 4. The Z*i*P implementation, however, is augmented with an optional Cancel flow from the seller to the buyer after the Payment flow.

The combined protocol flags set jointly by Buyer and Seller are listed in Figure 8.

The seller may choose (PFLAGS:CLRN) to combine payment authorization with payment clearing in which case the present protocol suffices. Alternatively, the seller may decide to only authorize payment and perform a separate clearance/capture function (described below) at some later time.

If requested by the buyer (PFLAGS:CONFIRM), the seller sends a Confirm to the buyer (optionally containing Sig_A if PFLAGS:SIG_A was set) regardless of the acquirer's decision (positive or negative) in Auth-Response.

If the seller chooses to (or is forced to) delay contacting the acquirer, he can send a Status flow (see below) to the buyer after receiving Payment. This is to keep the buyer abreast of the transaction status. Alternatively, the seller can elect to take the risk and send a Confirm to the buyer without having any real contact with the acquirer.

In the event that the seller is unable or unwilling, for some reason, to process the buyer's payment, the payment authorization protocol may be truncated (terminated) with a Cancel flow before trying to contact the acquirer.

The payment authorization protocol may also be suspended by the Buyer after the second (Invoice) flow. This abbreviated version can be used for gathering Sig_S's (i.e., signed invoices) from multiple sellers for the purpose of browsing and comparative shopping. The abbreviated protocol run can be resumed at a later time provided that Sig_S is still timely/valid.

SEPARATE PAYMENT CLEARANCE/CAPTURE. The protocol is shown in Figure 9. At the discretion of the seller, payment clearance may be performed either as part of authorization (described above) or postponed until later. This protocol supports delayed/separate clearance. (Of course, an acquirer may dictate its policy on this subject to all constituent sellers.)

[6]Note that most other micro-payment protocols such as Millicent [GMA+95] and NetBill [CTS95] gain their efficiency through the use of shared-key cryptosystems and therefore require complete trust in the payment system provider

[7]The terms "clearance" and "capture" are used interchangeably throughout this document.

PFLAGS:SIG_B	Buyer's signature Sig_B in Payment and Auth-Request. While this option is at the discretion of the buyer, a seller can refuse to issue an Invoice if it is the seller's policy to always require Sig_B and the buyer is not able to provide it. IMPORTANT: PFLAGS:SIG_B must be fixed for a given buyer-account combination. In other words, a buyer who has the ability to generate signatures must always do so. However, it is ultimately the acquirer's responsibility to make sure that a buyer with a signature capability always uses PFLAGS:SIG_B.
PFLAGS:SIG_S	Sellers's signature Sig_S in Invoice. This option is set by the buyer but, as before, a given seller can refuse to comply because, for example, it is not interested in giving out signed "offers" for buyers that aren't ready to pay.
PFLAGS:CONFIRM	Indicates that Confirm is requested. It is set by the buyer; it is envisaged that every seller should support this option.
PFLAGS:SIG_A	Sig_A is requested in Confirm. It is set by the buyer. NOTE: this option can only be used in conjunction with the PFLAGS:CONFIRM option.
PFLAGS:CLRN	Authorization and clearance (capture) are performed together. This option is set by the seller. While a buyer may, in principle, refuse it, it is not likely.
PFLAGS:noEnc	The buyer does NOT use encryption; SLIP is sent in the clear. This flag is set by the buyer. Sellers have no say over this option. The purpose of this option is to avoid the expense of encryption and to satisfy certain export regulations in cases when BAN's are not treated as secret or sensitive information. NOTE: this option can only be used in conjunction with the PFLAGS:SIG_B option.

Figure 8: *Protocol flags used in Z*i*P*

Multiple clearance flows against the same payment authorization are supported.

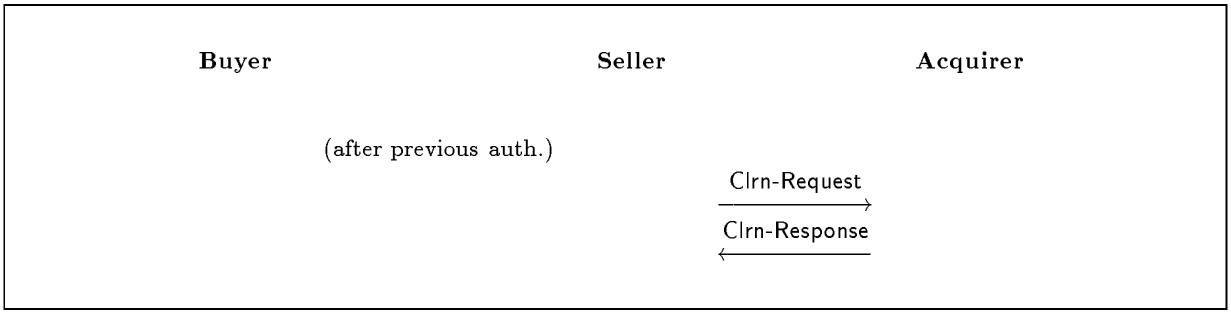

Figure 9: *The payment clearance/capture scenario*

REFUNDS. Sellers may issue refunds for previously cleared payments. Although it is understood that refunds are typically triggered by consumers/buyers, the interaction between buyer and seller that leads to an eventual refund is assumed to take place off-line (i.e., outside *i*KP /Z*i*P.)

Within *i*KP /Z*i*P, a refund transaction—for all practical purposes—is equivalent to (and treated as) a clearance/capture transaction. This is mainly because a refund is, essentially, a clearance with the lower amount. The difference between a refund and a clearance manifests itself only within the domain of the financial clearing network.

INQUIRY. The buyer can ask the seller about the status of a specific payment. The protocol is shown in Figure 10. The buyer may transmit Inquiry at any time after submitting a Payment flow. The seller must be able to respond for some time after the payment transaction is completed; the exact time period is the choice of the seller or may be specified by the financial institutions.

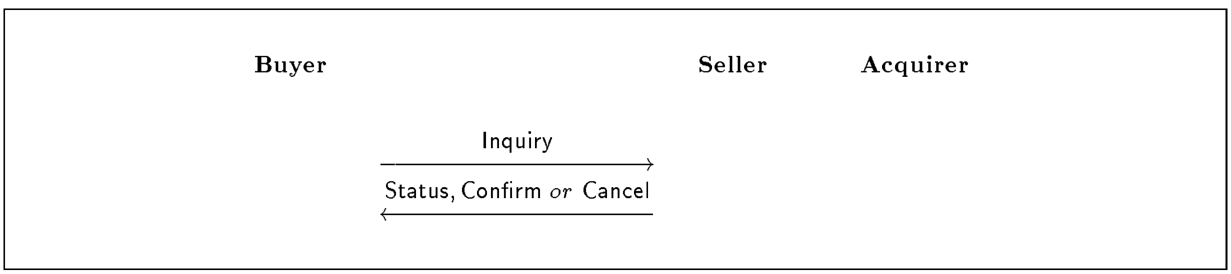

Figure 10: *The status inquiry scenario*

6.2 Implementation rationales and explanations

This section explains some of the features of the Z*i*P protocol design. For the detailed Z*i*P protocol specifications, we refer to Appendix A.

OPAQUE FIELDS. Some protocol fields are treated opaquely by Z*i*P. "Opaque" in this context means that these flows are not carried within the protocol messages. At the same time, these fields are authenticated and integrity-protected by Z*i*P. Some of these fields may (and sometimes have to) be tacked on by the higher-layer software. These fields are:

- DESC. Purchase details (e.g., merchandise description) may have to be explicitly transmitted between buyers and sellers. However, it is not recommended for transmission to the acquirer.
- TID_S and TID_B. Seller's and buyer's transaction identifiers are generated outside of Z*i*P (by the higher-layer software). By virtue of being part of Common they are integrity-protected by Z*i*P. Details of their

transport is left unspecified. Since both values serve as input to the hash $\mathcal{H}$(Common), association management and replay detection by the Transaction Layer (see Section 6.3) will largely be based on $\mathcal{H}$(Common). (An exception is Initiate: since this flow doesn't contain $\mathcal{H}$(Common), TID_B is used as the replay detection key for this flow.)

– All fields of the form OPT-SIG_Z are (as the name suggests) optional. Like the Text_i fields in Section 5 they carry optional data; unlike Text_i, OPT-SIG_Z fields are included in the respective Sig_Z signatures. For example, they can be used to carry credentials/certificates of various entities (hence their absence from the $Z\!i$P protocol flows), or other optional data like periodic account statements.

RECEIPT FROM SELLER. The protocols in Section 5 only have one "committing" value V whereby the Seller commits to whatever outcome is signaled by the Acquirer in Sig_A.

In $Z\!i$P, there are two different "committing" values: V (used for positive Confirm messages) and VC (negative Confirm and Cancel). Thus, the combination of Sig_S and V provides the buyer with an undeniable receipt of the payment; while the combination of Sig_S and VC is a proof for the buyer that the seller committed to never capturing this payment. Note that by using different values (V, VC) for positive and negative confirms (as opposed to the 2KP/3KP protocols in Section 5), these values now commit the Seller to a unique transacton outcome without linking with a Sig_A. This allows for a cleaner "decoupling" of the payment receipt (Sig_S, V or Sig_S, VC) by the Seller from the transaction authorization (Sig_A) by the Acquirer. It also explains why a Confirm(PFLAGS:CONFIRM) without Sig_A (PFLAGS:SIG_A) does have a value as a receipt.)

TYPING OF MESSAGE FIELDS. Every signature type generated in $Z\!i$P is assigned a unique signature identifier. Every signature operation (generation/verification) automatically includes a signature identifier for a specific signature type. The same holds for all hash function computations.

The following distinct signature types are identified: Sig_B, Sig_S, Sig_A, SigClrn_S, SigClrn_A.[8]

The following hash function computations are uniquely identified:$\mathcal{H}$(Common), $\mathcal{H}(V)$, $\mathcal{H}(VC)$, $\mathcal{H}_k(R_B$, BAN) and $\mathcal{H}_k(\text{SALT}_B, \text{DESC})$.

PERFORMANCE. Cryptographic operations such as computation/verification of public key signatures and en/de-cryption are computationally expensive. iKP and $Z\!i$P are designed to improve performance by minimizing the number of cryptographic operations.

ADHERENCE TO EXPORT REGULATIONS. The protocol is designed to minimize the amount of data encrypted, in order to satisfy the export control rules of the U.S. government. As described in detail in A.3, only SLIP is encrypted, and the data therein is limited to financial information.

6.3 Architecture

Figure 11 shows the architecture of $Z\!i$P. Buyer, Seller and Acquirer applications, residing on different network nodes, access the iKP functionality through the Transaction Layer interface [Lar96]. The Transaction Layer provides the payment applications with a high-level (C++) interface using simple payment objects, such as a BuyerTransaction class with methods Initiate(), Pay(), Inquiry(). The Transaction Layer takes care of association management, audit and configuration. The assocation management finds transactions matching incoming messages based on transaction ids (or $H(COMMON)$, as described in Section 6.1). It detects duplicate messages and unexpected messages, i.e., those not corresponding to an outstanding or recorded transaction. Unexpected requests are acknowledged with an error message, unexpected replays are ignored.

The Transaction Layer realizes robustness of the $Z\!i$P protocols in the presence of failures. It keeps transaction state in persistant storage and recovers from system crashes. To overcome unreliable communication, the Transaction Layer tries retransmitting after appropriate timeouts. The replay detection in the Transaction Layer will catch this and correspondingly resend the previous reply. (Note that the messages are idempotent!)

The Comm module sends iKP messages between different $Z\!i$P users. It supports several underlying transport mechanisms such as HTTP, Internet e-mail, TCP/IP.

[8] Since multiple clearance transactions against the same payment authorization are allowed, the clearance signatures (SigClrn_S, SigClrn_A) are further distinguished by CLRN-SEQ.

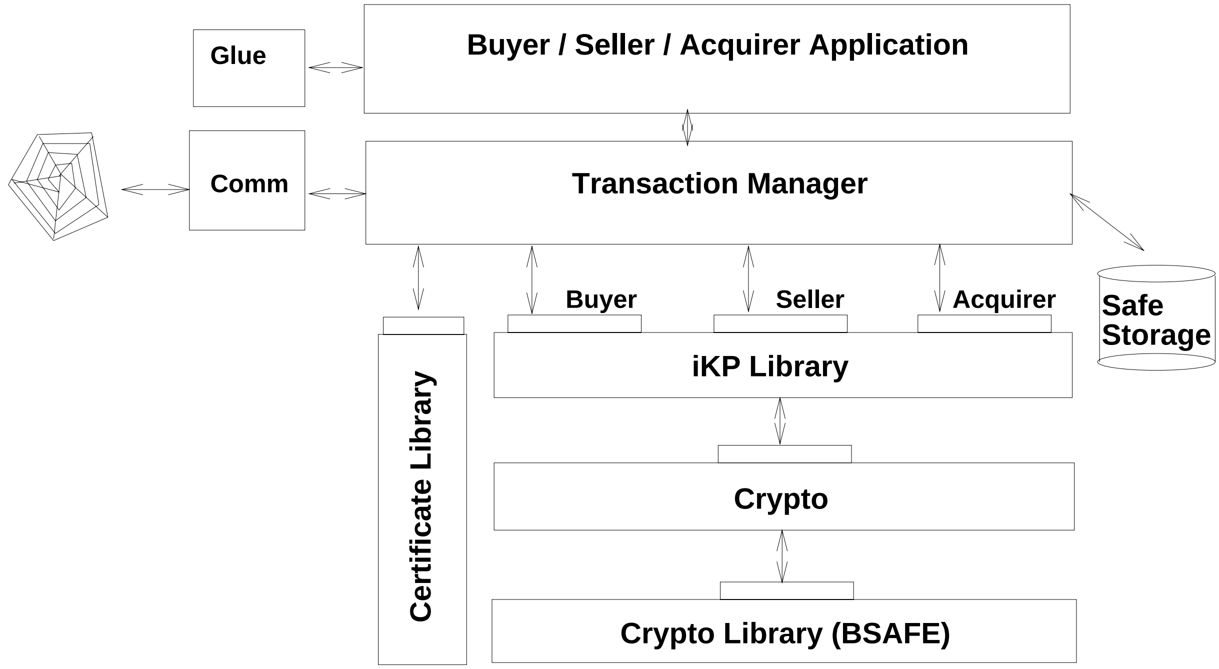

Figure 11: Z*i*P *implementation architecture*

Glue is an optional part that glues the network-independent buyer application to a user application such as a WWW browser, a CD-ROM catalog, etc. (see [HS95])

Payment applications, Transaction Layer, Glue, Comm and Safe Storage are all implemented in C++. The lower layers, consisting of the *i*KP, certificate and crypto libraries, are written in C.

The iKP Library [Tsu96] provides the core functionality for composing and verifying *i*KP protocol messages, using Certificate Library [Van96] for verifying certificates and Crypto [Ste96] for accessing cryptographic primitives. The iKP Library allows Buyer, Seller and Acquirer applications to verify received *i*KP messages against a context kept by the Transaction Layer, and to compose *i*KP messages to be sent. It returns an updated state to the Transaction Layer after each successful verification or composition.

Crypto separates the *i*KP protocol functionality from the cryptographic functionality, and allows support for different cryptographic toolkits. The Crypto API consists of methods for signature generation and verification, encryption and decryption, hashing and key handling (generation, destruction). The Z*i*P Crypto module is based on the RSA and MD5 functions of the RSA BSAFE 2.1 library. It provides additional functionality such as the randomized and plaintext-aware encryption (using OAEP) described in Appendix A.3. It also provides an interface for seeding the random number generator and implements seed collection combining various sources of randomness such as network traffic and inter-keystroke intervals of user-provided data.

Z*i*P also implements a *dummy* Crypto module which allows for non-export-controlled and platform-independent testing, and serves as an example for implementors of new crypto modules.

The Certificate Library implements a simple certificate issuing and verification functionality for certificates and certificate chains in a hierarchical model. The content of the certificates is taken from the X.509 [ISO94] specification.

For a complete set of ZiP documentation, see `http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP_overview.html`.

6.4 Deployment

At Europay's Annual Members' Meeting in Seville, Spain, in June 1996, Europay and IBM jointly ran a small-scale trial allowing visitors to the conference to use their pre-loaded Europay CLIP purse card to make secure internet payments from a card reader-equipped terminal. The payment scheme used was an integration of CLIP card payment functionality with Z*i*P-3KP, resulting in a secure scheme for Internet payments from pre-loaded purses.

From April 1997 till February 1998, the EMP (Electronic Market Place) project in Japan, funded by MITI (Ministry of International Trade and Industry), deployed Z*i*P-3KP in a trial with 5 on-line merchants and 2000 users. Each user received a smartcard storing his/her Z*i*P account (keys and certificates), allowing secure Internet purchases from public kiosks and terminals.

Z*i*P is also the payment technology behind the I-Pay payment product offered by Interpay Nederland and the Dutch banks. I-Pay was launched as a trial in June 1996, offering debit-type purchases from initially twenty on-line shops, using Z*i*P-3KP. Later, Eurocard/Mastercard credit card payments were added to the I-Pay brand. Currently I-Pay is accepted by 80 on-line merchants and has a user base of 17000 users. In line with initial plans to use *i*KP only for the initial trial and to move to the more standardized SET [MV97] technology once available and accepted, a phased transition is currently replacing Z*i*P technology with SET technology.

Acknowledgements

We thank Phil Janson and Mark Linehan for helpful discussions, and Jose L. Abad Peiro, Hans Granqvist and Steen Larsen for their contributions to the implementation of *i*KP /Z*i*P.

References

[AJSW97] N. Asokan, Phil Janson, Michael Steiner, and Michael Waidner. State of the art in electronic payment systems. *IEEE Computer*, 30(9):28–35, September 1997. A Japanese translation of the article appeared in pp 195-201, *Nikkei Computer* (http://nc.nikkeibp.co.jp/jp/) issue of March 30, 1998.

[And94] Ross Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–41, November 1994.

[Aso98] N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, May 1998.

[ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Research in Security and Privacy, pages 86–99, Oakland, CA, May 1998. IEEE Computer Society Press. A minor bug in the proceedings version was fixed. An errata sheet, distributed at the conference, is available at http://www.zurich.ibm.com/Technology/Security/publications/1998/ASW98-errata.ps.gz; A related paper [?] is available as well.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO '96*, number 1109 in Lecture Notes in Computer Science, pages 1–15. Springer-Verlag, Berlin Germany, 1996.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption – how to encrypt with rsa. In I.B. Damgard, editor, *Advances in Cryptology – EUROCRYPT '94*, Lecture Notes in Computer Science, pages 92–111. Springer-Verlag, Berlin Germany, 1994. final (revised) version appeared November 19, 1995.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security – Repelling the Wily Hacker*. Professional Computing Series. Addison-Wesley, 1994. ISBN 0-201-63357-4.

[CD97] Eui-Suk Chung and Daniel Dardailler. Joint electronic payment initiative (jepi). White paper, JEPI, April 1997.

[CGHK98] Pau-Chen Cheng, Juan Garay, Amir Herzberg, and Hugo Krawczyk. A security architecture for the internet protocol. *IBM Systems Journal, Special issue on the Internet*, 37(1):42–60, 1998. Updated version of [?].

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO '98*, number 1462 in Lecture Notes in Computer Science, pages 13–25. Springer-Verlag, Berlin Germany, August 1998.

[CTS95] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. NetBill security and transaction protocol. In *First USENIX Workshop on Electronic Commerce* [USE95].

[DA98] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. Internet Draft, November 1998. Expires May 12, 1999.

[FKK96] Alan O. Freier, Philip Kariton, and Paul C. Kocher. The SSL protocol: Version 3.0. Technical report, Internet Draft, 1996. Will be eventually replaced by TLS.

[GMA+95] Steve Glassman, Mark Manasse, Martin Abadi, Paul Gauthier, and Patrick Sobalvarro. The millicent protocol for inexpensive electronic commerce. In *Fourth International Conference on the World-Wide Web*, MIT, Boston, December 1995.

[HS95] Ralf Hauser and Michael Steiner. Generic extensions of WWW browsers. In *First USENIX Workshop on Electronic Commerce* [USE95], pages 147–154.

[HSW96] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-payments based on iKP. Research Report 2791 (# 89269), IBM Research, February 1996.

[ISO94] ISO/IEC. Information technology - open systems interconnection - the directory: Authentication framework, June 1994. same as ITU-T Rec X.509.

[Kra99] Hugo Krawczyk. Blinding of credit card numbers in the SET protocol. In *Proceedings of the 3rd Conference on Financial Cryptography (FC '99)*, Anguilla, British West Indies, Feb 1999. International Financial Cryptography Association (IFCA).

[Lar96] Steen Larsen. *Zurich iKP Prototype (ZiP): iKP Transaction Layer Functional Specification.* IBM Zurich Research Laboratory, May 1996.

[MV97] Mastercard and Visa. *SET Secure Electronic Transactions Protocol*, version 1.0 edition, May 1997. Book One: Business Specifications, Book Two: Technical Specification, Book Three: Formal Protocol Definition. Available from http://www.setco.org/set_specifications.html.

[NIS95] NIST National Institute of Standards and Technology (Computer Systems Laboratory). Secure hash standard. Federal Information Processing Standards Publication FIPS PUB 180-1, April 1995.

[Riv92] Ron Rivest. The MD5 message-digest algorithm. Internet RFC 1321, April 1992.

[Ste96] Michael Steiner. *Zurich iKP Prototype (ZiP): Cryptographic Library Specification.* IBM Zurich Research Laboratory, March 1996.

[Tsu96] Gene Tsudik. Zürich iKP prototype: Protocol specification document. Research Report RZ 2792, IBM Research, February 1996.

[USE95] USENIX. *First USENIX Workshop on Electronic Commerce*, New York, July 1995.

[Van96] Els Van Herreweghen. *Zurich iKP Prototype (ZiP): Certificate Library (CERT) Specification.* IBM Zurich Research Laboratory, February 1996.

[Wai96] Michael Waidner. Development of a secure electronic marketplace for Europe. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proceedings of the Fourth European Symposium on Research in Computer Security (ESORICS)*, number 1146 in Lecture Notes in Computer Science, Rome, Italy, September 1996. Springer-Verlag, Berlin Germany. also published in: EDI Forum 9/2 (1996) 98-106, see also `http://www.semper.org`.

A Detailed Z*i*P Protocol Description

A.1 Detailed protocol description

Fields and symbols. Figure 12 describes the (atomic) fields and cryptographic keys that are used in Z*i*P, in addition to those described in Figures 2 and 3. The composite fields/symbols are described in the protocol figures. Figure 12 reflects the addition or renaming of fields caused by the additional scenarios included in Z*i*P as opposed to the basic *i*KP flows. Examples of renaming are AUTH-PRICE (*i*KP: was PRICE) to be distinguished from CLRN-PRICE needed in the separate clearance scenario; or RESP-CODE (*i*KP: was Y/N) reflecting the non-binary nature (authorization denied, authorization approved but payment not captured, authorization approved and payment captured) of the Acquirer's response. The timestamps (AUTH-TIME, CLRN-TIME, INVOICE-EXP) were added to Z*i*P to enable efficient replay detection. Adding EXPIRATION to SLIP was required for reasons of compatibility with existing clearing practices. TID_B was included in the Common structure to allow for input by the Buyer in the set of transaction identifiers (TID_S, TID_B).

PFLAGS	Protocol option flags (see Figure 8).
VC	Random number generated by the seller, used to bind negative Confirm/Cancel and Invoice messages.
AUTH-PRICE	Amount and currency code authorized.
CLRN-PRICE	Amount and currency code cleared; may differ from AUTH-PRICE.
AUTH-TIME	Timestamp of payment authorization; set by acquirer
CLRN-TIME	Timestamp of payment clearance; set by acquirer
CLRN-SEQ	Clearance sequence number set by seller. Initially set to 0 and incremented for each successive clearance/refund.
SALT_C	Random number used to salt the account number in the buyer's certificate.
EXPIRATION	Expiration date of the buyer's account.
INVOICE-EXP	Invoice (offer) expiration specified by the seller.
RESP-CODE	Authorization or capture response code from the acquirer. Can also be set by the seller in case of a cancellation.
TID_Z	Transaction id assigned be each party to an *i*KP transaction. Generated at a layer above *i*KP and not explicitly carried in *i*KP flows (higher layers transport TIDs).
OPT-SIG_Z	Optional text for inclusion in signatures. Not explicitly carried in iKP flows.

Figure 12: *Atomic fields and symbols used in* Z*i*P

Payment with in-line authorization. The basic payment authorization protocol of Z*i*P is shown in Figure 13. PFLAGS:CLRN (in Clear) indicates whether the seller wishes the acquirer to perform payment clearance at the same time. The response code from the acquirer indicates whether authorization is given, and (if clearance was requested) whether the payment was cleared.

- **Composite Fields:**

Common	TID_S, TID_B, PFLAGS, AUTH-PRICE, ID_S, DATE, INVOICE-EXP, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, $\mathcal{H}(V)$, $\mathcal{H}(VC)$
Clear	PFLAGS, ID_S, DATE, NONCE_S, $\mathcal{H}$(Common), INVOICE-EXP, $\mathcal{H}(V)$, $\mathcal{H}(VC)$
SLIP	AUTH – PRICE, $\mathcal{H}$(Common), BAN, R_B, EXPIRATION, [SALT_C or PIN]
EncSlip	$\mathcal{E}_A$(SLIP)
Sig_S	$\mathcal{S}_S(\mathcal{H}(\mathrm{Common}), \mathrm{OPT\text{-}SIG}_S)$
Sig_B	$\mathcal{S}_B(\mathrm{EncSlip}, \mathcal{H}(\mathrm{Common}), \mathrm{OPT\text{-}SIG}_B)$
Sig_A	$\mathcal{S}_A(\mathrm{RESP\text{-}CODE}, \mathrm{AUTH\text{-}TIME}, \mathcal{H}(\mathrm{Common}), \mathrm{OPT\text{-}SIG}_A)$

- **Protocol Flows:**

Initiate:	B	PFLAGS, SALT_B, ID_B $\longrightarrow$	S
Invoice:	B	$\longleftarrow$ Clear, [Sig_S]	S
Payment:	B	EncSlip, [Sig_B] $\longrightarrow$	S
Cancel:	B	$\longleftarrow$ RESP-CODE, VC, [Sig_S]	S
Auth-Request:	S	Clear, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, EncSlip, Sig_S, [Sig_B,] $\longrightarrow$	A
Auth-Response:	S	$\longleftarrow$ RESP-CODE, AUTH-TIME, Sig_A	A
Confirm:	B	$\longleftarrow$ RESP-CODE(=positive), AUTH-TIME, V, [Sig_S], [Sig_A]	S
		—OR—	
Confirm:	B	$\longleftarrow$ RESP-CODE(=negative), AUTH-TIME, VC, [Sig_S], [Sig_A]	S

Figure 13: ZiP: *Payment with in-line authorization*

Note that the Cancel flow is only used if the seller decides—for whatever reason—not to go ahead with payment authorization. Cancel may not be used if the seller has received an Auth-Response.

If the payment clearance is not done (either because it was not requested by the seller, or because the acquirer was unable to perform it), the seller must subsequently either transmit the Clrn-Request message or take further processing of this payment off-line.

Clearance. The clearance flows of $\mathcal{Z}i$P are shown in Figure 14. The seller uses this protocol to request the actual transfer of funds from the buyer to the seller. This flow may only occur after the corresponding authorization flow if clearance was not performed as part of payment authorization. The seller must send the Clrn-Request message to the same acquirer used for payment authorization.

Inquiry. The flows of $\mathcal{Z}i$P's Inquiry protocol are shown in Figure 15. The buyer may transmit Inquiry at any time after sending the Payment flow. The response from the seller can be either Confirm (if the seller has received Auth-Response), a Status message with his view on the current transaction state (if he hasn't received Auth-Response), or Cancel (if he decided to cancel the payment and hasn't previously sent Auth-Request to the acquirer nor Confirm to the buyer).

Protocol options and flags. The payment authorization protocol may be suspended after the second (Invoice) flow. This abbreviated version can be used for gathering Sig_S's (i.e., signed invoices) from multiple sellers for the purpose of browsing and comparative shopping. The abbreviated protocol run can be resumed at a later time provided that Sig_S is still timely/valid.

In addition, $\mathcal{Z}i$P supports the transaction options listed in Figure 8.

A.2 Protocol flow processing

This section describes, step by step, the normal protocol operation by all parties. The handling of errors and other exceptions by the Transaction Layeris discussed in Section 6.3discussed in the next section.

All parties involved are assumed to have access to stable, non-volatile storage. The term "recording" means commitment to stable storage. It is further assumed that the buyer and the seller commit all local variables—transaction ids, nonces, signatures, etc.—to stable storage *before* sending out a message flow. Moreover, it is assumed below that incoming flows are associated (matched) with outstanding, currently-active transaction by the transaction layer software, i.e., software residing above the $\mathcal{Z}i$P message processing code.

A.2.1 Payment authorization protocol.

This section discusses how basic payment authorization is handled by all participants.

- **Composite Fields:**

Common	TID_S, TID_B, PFLAGS, AUTH-PRICE, ID_S, DATE, INVOICE-EXP, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B,\mathrm{DESC})$, $\mathcal{H}(V)$, $\mathcal{H}(VC)$
$\mathrm{SigClrn}_S$	$\mathcal{S}_S(\mathcal{H}(\mathrm{Common}),$ CLRN-PRICE, OPT-SIG-CLRN$_S$, CLRN-SEQ)
$\mathrm{SigClrn}_A$	$\mathcal{S}_A($RESP-CODE, CLRN-TIME, CLRN-PRICE, CLRN-SEQ, $\mathcal{H}(\mathrm{Common})$, OPT-SIG-CLRN$_A$)

- **Protocol Flows:**

Clrn-Request: $\mathrm{S} \xrightarrow{\mathcal{H}(\mathrm{Common}),\text{CLRN-PRICE, CLRN-SEQ, }\mathrm{SigClrn}_S} \mathrm{A}$

Clrn-Response: $\mathrm{S} \xleftarrow{\text{RESP-CODE, CLRN-TIME, CLRN-SEQ, }\mathrm{SigClrn}_A} \mathrm{A}$

Figure 14: $\mathcal{Z}i$P*: Clearance protocol*

- **Composite Fields:**

Common	TID_S, TID_B, PFLAGS, AUTH-PRICE, ID_S, DATE, INVOICE-EXP, NONCE_S, ID_B, $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$, $\mathcal{H}(V)$, $\mathcal{H}(VC)$

- **Protocol Flows:**

Inquiry: B $\xrightarrow{\mathcal{H}(\text{Common})}$ S

Confirm: S $\xleftarrow{\text{(same as in Figure 13)}}$ S

—OR—

Cancel: S $\xleftarrow{\text{(same as in Figure 13)}}$ S

—OR—

Status: S $\xleftarrow{\text{STATE}}$ S

Figure 15: Z*i*P: *Inquiry protocol*

Initiate COMPOSITION. The buyer transmits the Initiate message at the end of the negotiation phase and the start of the payment phase; this message delimits the boundary between the two phases.

Note that DESC (purchase details) and AUTH-PRICE (purchase amount) are either agreed upon beforehand or communicated in-band alongside the Initiate or Invoice flows. In any case, DESC is not carried within any Z*i*P flow. The buyer forms INITIATE as follows:

1. Produces ID_B by generating random number R_B and computing $\mathrm{ID}_B = \mathcal{H}_k(R_B, \mathrm{BAN})$.
2. Generates another random number SALT_B to be used for "salting" the hash of the purchase description (DESC) in subsequent messages; also used as a challenge to the seller.
3. Sets PFLAGS to reflect desired protocol options.
4. Sends Initiate.

Initiate PROCESSING AND Invoice COMPOSITION. Upon receipt of the Initiate flow, the seller performs the following steps:

1. Checks buyer options set in PFLAGS and returns an error if it finds it incompatible with its own policy. Otherwise, the seller sets the PFLAGS:CLRN option flag in PFLAGS to reflect its payment processing policy.
2. Gets a clock reading and sets DATE.
3. Generates nonce NONCE_S to be used later for freshness/uniqueness checks.
4. Computes $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$.
5. Generates random $[V, VC]$ pair and computes the corresponding $[\mathcal{H}(V), \mathcal{H}(VC)]$ vector.
6. Encodes/linearizes Common as defined above and computes $\mathcal{H}$(Common).
   NOTE: The seller does not need to additionally "salt" $\mathcal{H}$(Common) because it contains the already-salted $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$ as well as NONCE_S.
7. Computes Sig_S if the PFLAGS:SIG_S flag is set.
8. sends Invoice.

Invoice PROCESSING. Upon receiving Invoice the buyer takes the following steps:

1. Retrieves the fields in Clear.

2. Validates DATE (within an implementation-defined skew).
3. Computes $\mathcal{H}$(Common) and compares it with $\mathcal{H}$(Common) from Clear. This confirms that the buyer and seller agree on information contained in Clear, in particular AUTH-PRICE and DESC (through $\mathcal{H}_k(\mathrm{SALT}_B, \mathrm{DESC})$).
4. Verifies Sig_S if the PFLAGS:SIG_S option is set.

The protocol may terminate at this point since, sometimes, the buyer may choose not to proceed with the actual payment. The seller keeps transaction state for some implementation-specific period of time (perhaps set to the maximum time skew for a given acquirer.) Thereafter, the transaction is deleted.

Payment COMPOSITION. The buyer performs the following steps when it is ready to proceed with the payment:
1. If the PFLAGS:noEnc flag is not set, forms SLIP and encrypts it under PKE_A as described in "Use of encryption in *i*KP /Z*i*P" below. Include SALT_C in SLIP if the PFLAGS:SIG_B flag is set.
2. Otherwise, SLIP is simply linearized (encoded) in the clear. Note that in this case neither PIN nor SALT_C are set in SLIP.
3. If PFLAGS:SIG_B is set, computes Sig_B.
4. Sends Payment.

Payment PROCESSING. Upon receiving Payment, the seller first does the following:
1. Verifies Sig_B if PFLAGS:SIG_B set and the buyer's credentials are available.
   NOTE: If buyer's credentials are not available, the seller may decide to either cancel the transaction or go ahead with it. In case of the former, a Cancel message is sent to the buyer.
2. Chooses whether to perform immediate or delayed authorization. If the former, proceeds to Auth-Request message processing. Otherwise, seller returns Status to the buyer.
3. If, for any reason, the seller decides not to process the payment further, it can generate and return a Cancel message to the buyer.

Auth-Request COMPOSITION. The seller performs the following steps when it is ready to obtain payment authorization:
1. If Sig_S was computed for Invoice, copies the same Sig_S into Auth-Request.
2. If Sig_S is not already computed, computes Sig_S and places it in Auth-Request.
3. Sends Auth-Request.

Auth-Request PROCESSING. When the acquirer receives Auth-Request, it:
1. Checks that the DATE is valid (within an implementation-defined skew).
2. Check for replays based on $\mathcal{H}$(Common). Reply with previous response if replay.
3. Verifies Sig_S.
4. Decrypts EncSlip and obtains:
   AUTH-PRICE, BAN, R_B, EXPIRATION, $\mathcal{H}$(Common) and optionally PINand $salt_C$.
5. Verifies Sig_B if present. This includes computing $\mathcal{H}_k(\mathrm{SALT}_C, \mathrm{BAN})$ using the previously decrypted values and matching it with the salted account number in the certificate CERT_B. Note that Sig_B must always be present if PFLAGS:SIG_B option flag is set and if the buyer has Sig_B capability.
6. Encodes/linearizes Common; then, computes $\mathcal{H}$(Common) and cross-checks it with both $\mathcal{H}$(Common) in Clear and $\mathcal{H}$(Common) in SLIP.
7. Validates that R_B and BAN match ID_B found in Clear.
8. Composes and sends authorization request on the financial network. If PFLAGS:CLRN was set, includes a capture request with the authorization request.
9. Proceeds to Auth-Response once the response is received from the financial network.

Note that the acquirer and/or financial network must perform replay detection only if the seller requests payment capture processing. This is to ensure that payments are not charged to buyers multiple times. Replay detection for authorization-only requests is a policy matter determined by the individual payment system providers.

Auth-Response COMPOSITION. When the acquirer receives an authorization (and possibly capture) response from the financial network, it:

1. Sets RESP-CODE to indicate any of:
   - Authorization denied.
   - Authorization approved but payment not captured.
   - Authorization approved and payment captured (only if capture requested in PFLAGS).
2. Sets $\mathrm{OPT}-\mathrm{SIG}_A$ to the authorization code and other optional data provided by the financial network. Data in $\mathrm{OPT}-\mathrm{SIG}_A$, while included in the computation of Sig_A, is treated opaquely by $Z\!i$P and is not carried in the $Z\!i$P messages. (Its transport is left up to the upper layer.)
3. Computes Sig_A.
4. Sends Auth-Response to the seller.

Auth-Response PROCESSING. When the seller receives Auth-Response, it:

1. Verifies Sig_A.
2. Records RESP-CODE, AUTH-TIME and $\mathrm{OPT}-\mathrm{SIG}_A$.
3. If RESP-CODE is positive, generates positive Confirm by including V in the message.
4. If RESP-CODE is negative, generates negative Confirm by including VC in the message.
5. Confirm is sent only if PFLAGS:CONFIRM flag is set.

Confirm COMPOSITION. The seller generates Confirm as follows:

1. Copies RESP-CODE and AUTH-TIME from Auth-Response.
2. If PFLAGS:SIG_A flag is set, copies Sig_A. (Note that $\mathrm{OPT}-\mathrm{SIG}_A$ has to be tacked on above $Z\!i$P.)
3. Copies Sig_S from Auth-Request (if not provided in Invoice).
4. Includes V (or VC) as evidence to the buyer that the payment transaction has completed.
5. Sends Confirm.

This completes the payment transaction from the seller's perspective. The only way the seller communicates with the buyer again about the same transaction is if the buyer sends Inquiry.

Confirm PROCESSING. When the buyer receives Confirm, it:

1. If PFLAGS:SIG_A flag is set, verifies Sig_A.
2. If PFLAGS:SIG_S flag is not set, verifies Sig_S.
3. Verifies that V hashes to $\mathcal{H}(V)$ (positive Confirm) or VC hashes to $\mathcal{H}(VC)$ (negative Confirm).
4. Records RESP-CODE, V, Sig_A, and Sig_S as evidence of the transaction's completion.

Cancel COMPOSITION. The seller generates Cancel as follows:

1. Sets RESP-CODE to a specific error code (e.g., merchandise out of stock.) Note that RESP-CODE in Cancel is set by the Seller and therefore its meaning is different from that in Auth-Response (and Confirm).
2. If Sig_S is not previously computed, computes Sig_S afresh.
3. Includes VC as evidence to the buyer that the payment transaction has been aborted, i.e., the seller promises not to pursue buyer's payment.
4. Sends Cancel.

Cancel PROCESSING. When the buyer receives Cancel, it:

1. If PFLAGS:SIG_S flag is not set, verifies Sig_S.
2. Verifies that VC hashes to $\mathcal{H}(VC)$.
3. Stores RESP-CODE, VC, and Sig_S as evidence of the transaction's dismissal.

Note that any Cancel received after a valid Confirm should be ignored.

Status PROCESSING. When the buyer receives Status, it:

1. Stores the current payment status (from the seller's perspective) reflected in STATE.

A.2.2 Clearance (Capture) protocol.

The seller utilizes the clearance protocol when doing on-line (rather than batch) payment capture processing, and the RESP-CODE received in Auth-Response indicates that payment is authorized but not captured. Note that clearance must be performed with the same acquirer that handled the previous authorization. Furthermore, clearance cannot be initiated before Auth-Response is received.

Moreover, as mentioned earlier, the clearance protocol may be used for implementing seller-initiated refunds.

Clrn-Request COMPOSITION. The seller performs the following steps:

1. Obtains CLRN-PRICE.
2. Computes SigClrn_S.
3. Increments CLRN-SEQ.
4. Sends Clrn-Request.

Clrn-Request PROCESSING. When the acquirer receives Clrn-Request, it:

1. Checks that the transaction is in the appropriate state, i.e., that either a) payment authorization or b) payment clearance, has been performed.
2. Examines previous clearance transactions against the same payment authorization and checks that CLRN-SEQ is new (i.e., greater than that in the last Clrn-Request processed.)
3. Verifies SigClrn_S.

Note that the acquirer and/or financial network *must* perform replay detection of capture requests. This is to ensure that payments are not charged to buyers multiple times.

The relationship between AUTH-PRICE and CLRN-PRICE (as well as that between CLRN-PRICE and earlier cleared amounts) is not checked within Z*i*P. The difference between the two values is subject to locally-defined constraints. (Checking is assumed to be done at the transaction layer or higher.)

Clrn-Response COMPOSITION. When the acquirer receives the appropriate response from the financial clearing network, it:

1. Sets RESP-CODE to indicate any of:
   - capture denied; or
   - capture completed.
2. Sets time-of-clearance, CLRN-TIME.
3. Computes SigClrn_A.
4. Sends Clrn-Response to the seller.

Clrn-Response PROCESSING. When the seller receives Clrn-Response, it:

1. Verifies SigClrn_A.
2. Archives RESP-CODE and other fields for use in any future Status or Confirm message.

A.2.3 Inquiry protocol.

Inquiry COMPOSITION. The buyer can issue Inquiry at any time after sending the Payment flow. Composing Inquiry does not require any special actions since its contents are limited to $\mathcal{H}$(Common).

Inquiry PROCESSING. When the seller receives an Inquiry flow he—depending on the current state of the transaction—transmits Confirm, Cancel or Status to the buyer.

A.3 The encryption function

*i*KP requires the buyer to encrypt the SLIP as part of the Payment message. (The sole exception to this is when both the PFLAGS:noEnc and the PFLAGS:SIG_B option flags are set.)

A.3.1 Payload

Encryption is *always* performed using the encryption public key of the acquirer–PKE_A—for the Payment message. It is assumed that PKE_A has a modulus size of at least 1024 bits. The format of the linearized (encoded) SLIP to be encrypted is as follows:

$$\text{SLIP} = [\text{AUTH} - \text{PRICE}, \mathcal{H}(\text{Common}), \text{BAN}, R_B, [\text{SALT}_C|\text{PIN}], \text{EXPIRATION}, \text{PADDING}]$$

Fields within SLIP are described in Figure 12. The sizes of these fields are given in Figure 16. All of the above components are encoded into a 896-bit long string.

AUTH-PRICE	64 bits
BAN	0-128 bits (128 bits can be used to encode up to 38 decimal digits. Current credit cards are only 12 digits.)
EXPIRATION	32 bits
SALT_C or PIN	0-64 bits (up to 19 decimal digits)
$\mathcal{H}(\text{Common})$	128 bits
R_B	At least 128 bits
PADDING	length is the difference between 832 bits and the sum (in bits) of all previous fields.

Figure 16: *Sizes of fields in* EncSlip

A.3.2 Encryption Process

The actual encryption process is adapted from [BR94]. The encryption function $\mathcal{E}_A$ is based on RSA. We let $f(x) = x^e \pmod N$ denote the RSA function and $f^{-1}(y) = y^d \pmod N$ its inverse, where N is a 1024 bit modulus. The issue is that simply encrypting under RSA– ie. setting $\mathcal{E}(x) = f(x)$– is not enough: this doesn't provide the "integrity" or "plaintext awareness" we need. Instead, we first "embed" a up to 832-bit plaintext into a 1024 bit string r in a very special way and then compute $f(r)$. The scheme we now describe is a simplification of a OAEP scheme from [BR94]. It makes use, in addition to RSA, of MD5 as the hash function $\mathcal{H}$, and is provably secure assuming $\mathcal{H}$ behaves like a "random function."

The encryption process is illustrated in Figure 17 and performs following steps:

1. Prepend 64 bits of zeros to 832 bits of DATA to form x=[0...64x...0, DATA].
2. Generate random 128-bit string E_SALT.
3. Compute a = x XOR H_1(E_SALT).
4. Let b = E_SALT XOR H_2(a).
5. Compute E_a(a,b) (combined length of a,b is 1024 bits).

`H_1` is a one-way function which expands data from one block of 128 bits to 896 bits and is illustrated in Figure 18.

`H_2` is a one-way function which compresses data of size 896 to a block of 128 bits. See Figure 19 for its implementation.

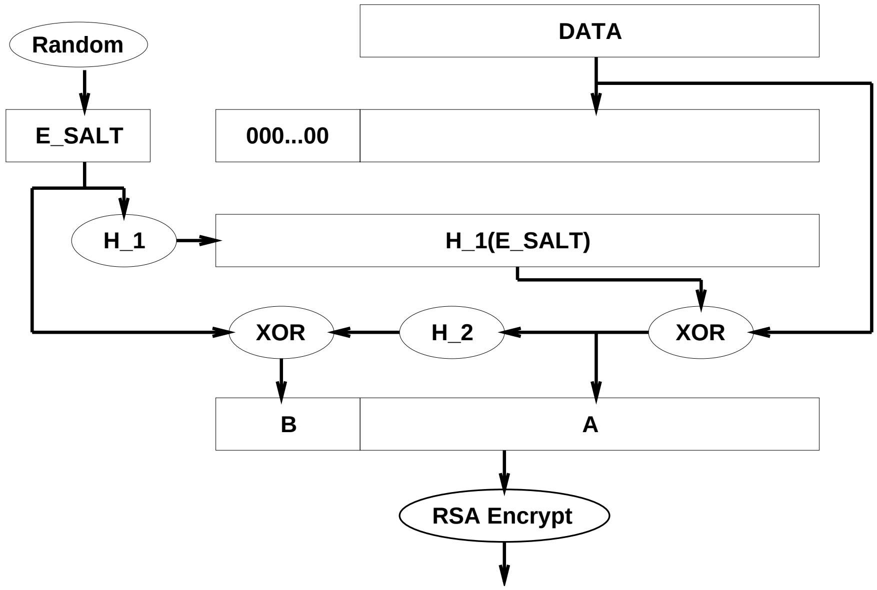

Figure 17: *Encryption using OAEP*

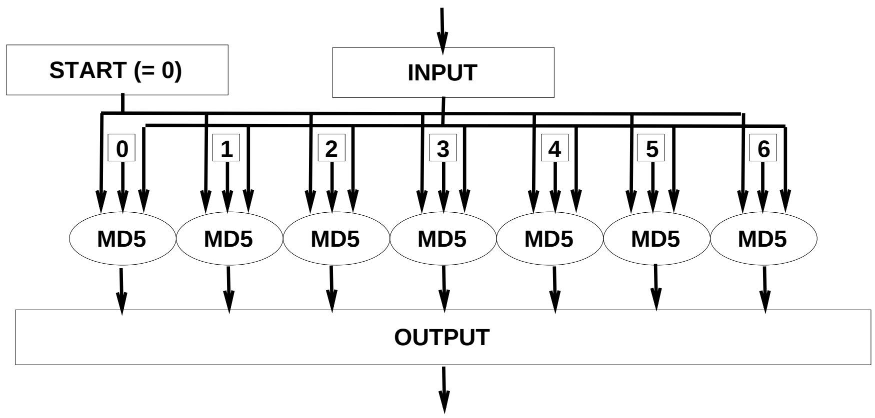

Figure 18: sl Hash-function H1 for OAEP

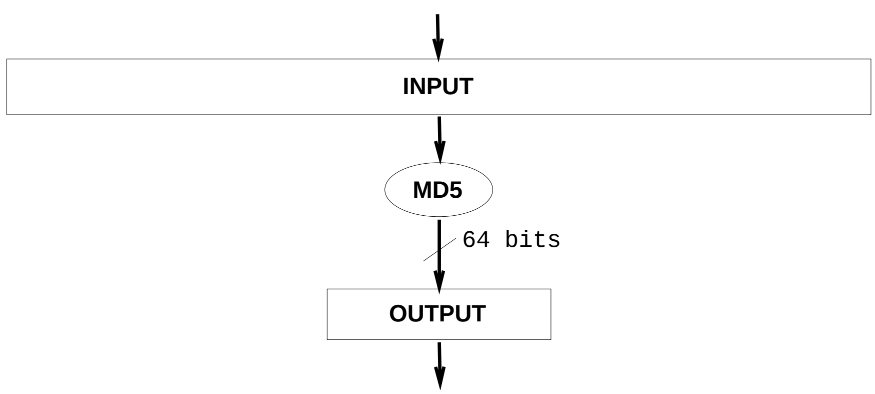

Figure 19: *Hash-function H2 for OAEP*

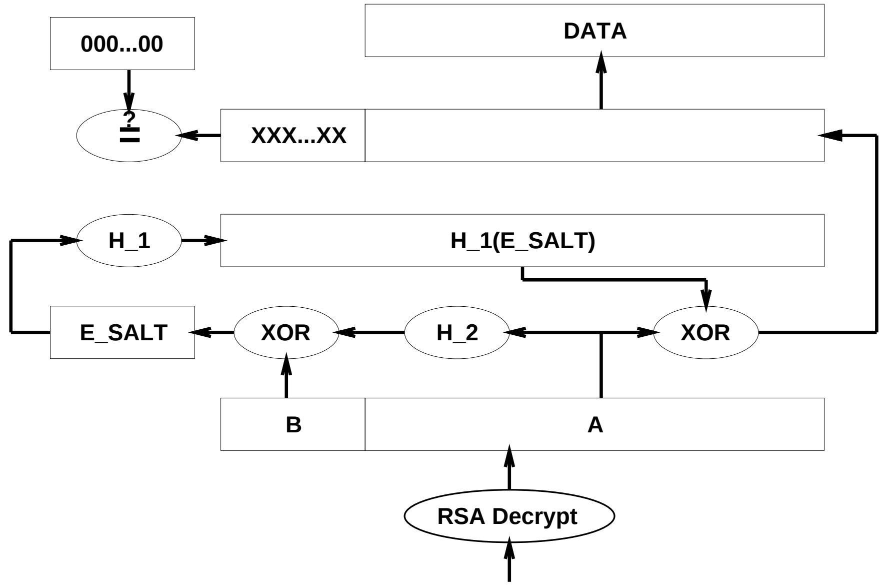

Figure 20: *Decryption using OAEP*

A.3.3 Decryption Process

The decryption process is illustrated in Figure 20 and performs the following steps:

1. Compute (a,b) = D_a(E_a(a,b)).
2. Compute E_SALT = b XOR H_2(a).
3. Compute x = a XOR H_1(E_SALT).
4. Check for existepnce of 64 leading 0-s in x and obtain DATA.